On-Demand Services SDK for VMware Tanzu v0.27

On-Demand Services SDK for VMware Tanzu 0.27

You can find the most up-to-date technical documentation on the VMware website at:
https://docs.vmware.com/

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

On-Demand Services SDK

On-Demand Services SDK

This guide is for people who want to author service tiles for Pivotal Cloud Foundry (PCF) using the on-demand services SDK, part of the Pivotal Cloud Foundry Services SDK.

Overview

PCF operators make software services such as databases available to developers by using the Ops Manager **Installation Dashboard** to install service tiles.

On-demand services let you provision instances in a flexible way. The operator does not pre-allocate a block of VMs for the instance pool, and they can specify an allowable range rather than fixed settings for instance resource levels. When a developer creates an on-demand service instance, they then provision it at creation time.

The on-demand services SDK provides a generic, on-demand broker (ODB). This simplifies broker and tile authoring, and is the standard approach for both Pivotal internal services teams and Pivotal partner independent software vendors (ISVs) to develop on-demand services for PCF. For more information about service brokers and how the on-demand broker works within PCF, see About On-Demand Brokers.

Product Snapshot

The following table provides version and version-support information about the on-demand services SDK.

Element	Details
Version	0.27.0
Release date	April 4, 2019
Compatible Ops Manager version(s)	2.0 or later
Compatible Pivotal Application Service (PAS) version(s)	2.0 or later
IaaS support	AWS, Azure, GCP, OpenStack, and vSphere

Key Features

The benefits of provisioning service instances with resources on-demand are:

- Operators can scale resource consumption in line with need, without having to plan for pre-provisioning.

- App developers get more control over resources and do not have to acquire them through the operator.

The benefits of using ODB to develop on-demand services are:

- ODB reduces the amount of code service developers have to write by abstracting away functionality common to most single-tenant on-demand service brokers.
- ODB uses BOSH to deploy service instances, so anything that is BOSH-deployable can integrate with Cloud Foundry' s services Marketplace.

ODB uses the following BOSH features:

- Dynamic IP management
- Availability zones
- Globally-defined resources (**Cloud Config**). This results in manifests that are portable across BOSH Cloud Provider Interfaces (CPIs) and are substantially smaller than old-style manifests.
- Links between deployed BOSH instances consuming information from other instances, for example, IP addresses.

Prerequisites for Deploying Brokers That Use ODB

For information about minimum versions of Cloud Foundry and BOSH, see Software Requirements.

Create a pull request or raise an issue on the source for this page in GitHub

On-Demand Services SDK Release Notes

Page last updated:

For product versions and upgrade paths, see Upgrade Planner.

v0.27.0

Release Date: April 4, 2019

Breaking Change: The On Demand Services SDK has replaced `ServiceDeployment.Stemcell` with `ServiceDeployment.Stemcells`. This change is to support multiple stemcells for a service deployment.
For more information, see [BREAKING CHANGE] SDK now supports multi-stemcell deployment in GitHub.

Breaking Change: To support bpm for the broker, you must include the bpm release in the broker job.

Also, because bpm restricts access to the current job, you must also make the broker signify to bpm that it needs access to the service adapter configuration.

To do so, specify the `mount_paths` to the service adapter within the `broker` job

`service_adapter` configuration. For an example of this configuration, see Starter Snippet for Your Broker.

Features

New feature in this release:

- The on-demand broker (ODB) now supports service instance deployments that use different stemcells for different instance groups.
- The broker binary now uses BOSH Process Manager (bpm) for better job isolation and security. For more information about bpm, see bpm in the BOSH documentation.

Known Issues

This release has the following issues:

- An issue with BOSH CredHub v1.9.11 and v2.1.5 causes an error after clicking **Apply Changes**. These versions of CredHub are included in Ops Manager v2.2.21, v2.3.15, v2.4.9, and v2.5.2.

  To avoid this error, use CredHub v1.9.12 or later (included in Ops Manager v2.2.22 or later and v2.3.16 or later) or CredHub v2.1.6 (included in Ops Manager v2.4.10 or later and v2.5.3 or later). For more information about this error, see Unable to Render Templates for Job CredHub.
- Contacting the BOSH Director links API many times in parallel can cause some requests to fail during operations such as bind and unbind.

Minimum Version Requirements

The following are minimum version requirements for this release:

- BOSH v266.12.0 or v267.6.0 and later
- Cloud Foundry v238 and later

For Pivotal Cloud Foundry (PCF) version support, see Product Snapshot.

View Release Notes for Another Version

To view the release notes for another product version, select the version from the dropdown at the top of this page.

Create a pull request or raise an issue on the source for this page in GitHub

About On-Demand Brokers

This topic provides information about on-demand brokers for people who want to create on-demand service tiles for Pivotal Cloud Foundry (PCF).

Cloud Foundry Service Brokers and PCF Tiles

Service brokers let developers create service instances in their development spaces that they can call from their code.

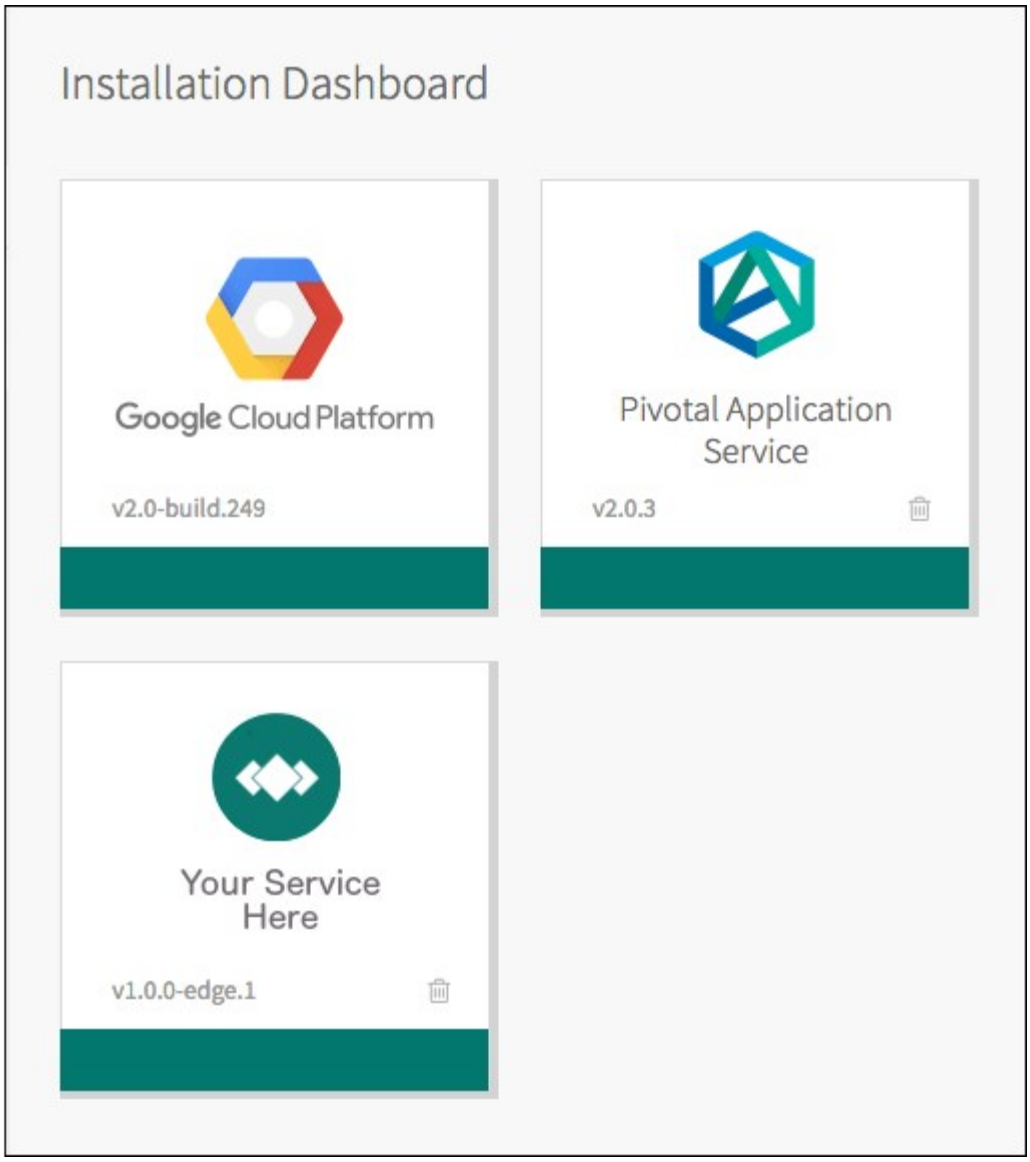
Cloud Foundry Service Brokers

Cloud Foundry Service Brokers provide an interface between the Cloud Controller and the add-on software service that they represent. The service broker works by providing an API which the Cloud Controller calls to create service instances, bind them to apps, and perform other operations. Cloud Foundry service brokers are implemented as HTTP servers that conform to the Open Service Broker API.

In addition to providing an API, a service broker publishes a service catalog that may include multiple service plans, such as a free tier and a metered tier. Brokers register their service plans with the Cloud Controller to populate the Marketplace, which developers access with `cf marketplace` or through the Pivotal Cloud Foundry (PCF) Apps Manager.

PCF Tiles

On PCF, operators find services on Pivotal Network and install and configure them through a tile interface in the Ops Manager **Installation Dashboard** to make them available to developers. Installing a service tile creates a service broker, registers it with the Cloud Controller, and publishes the service plans that the broker offers. Developers can then create service instances in their spaces and bind them to their apps.



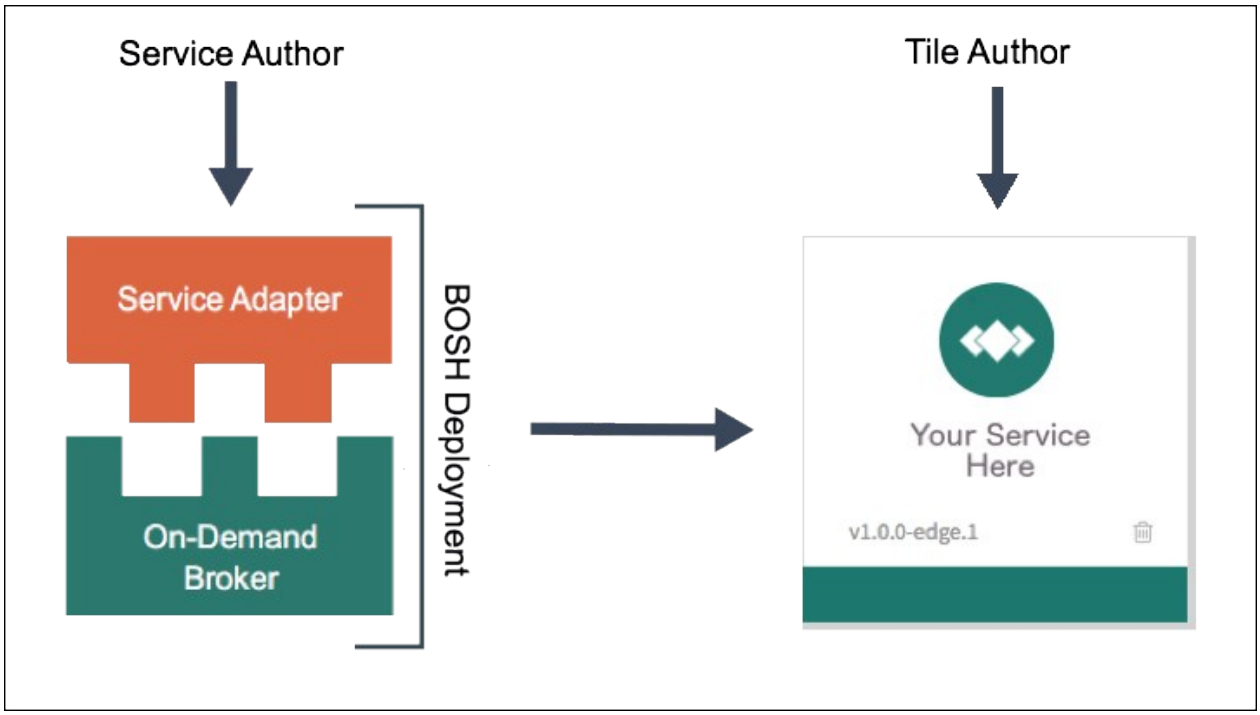
The central element behind a service tile is the service broker, but the tile software includes other components that make the service easy for operators to install and maintain and easy for developers to use. These components include configuration layouts, upgrade rules, lifecycle errands, and BOSH manifests for deploying the service instances.

On-Demand Services SDK and the On-Demand Broker

The on-demand services SDK provides a generic on-demand broker (ODB) that answers API calls from the Cloud Controller.

Service authors add service-specific functionality to the on-demand services SDK through an executable called a **Service Adapter**. These components combine to create a BOSH deployment. For more information about BOSH deployments, see What is a Deployment.

Tile authors customize the tile interface used by operators. The tile consumes the BOSH deployment to generate a BOSH manifest for deploying on-demand instances of the service. The diagram below illustrates this process.



The on-demand services SDK imposes no constraints on the service authors' ability to offer new functionality or expose configuration options in their service plans, such as rate limiting and external load balancers.

On-Demand Service Roles

There are several roles involved with creating and managing on-demand service tiles. These roles can be separate or combined. This section provides a summary of the responsibilities for each role.

Service Author

Service authors write and maintain the service adapter. For more information about service author responsibilities, see Service Author Deliverables.

Tile Author

Tile authors determine which configuration options to expose to Ops Manager operators, create the tile, and publish it on Pivotal Network. For more information, see Creating an On-Demand Service Tile.

Operator

Operators deploy and maintain the broker. They also manage access control for Cloud Foundry (CF) developers. This documentation provides information for two types of operator, Ops Manager operators and BOSH operators. These roles may be separate or combined. The following describes each type of operator:

- **Ops Manager Operator**: Uses the Ops Manager UI to configure plans and provide service specific configurations. For more information about configuring a specific service, see the Pivotal Documentation for the service.
- **BOSH Operator**: Creates and modifies the on-demand service broker manifest to provide

service specific configurations. For more information about operator responsibilities, see Operator Responsibilities.

Service Network Requirement

When you deploy Pivotal Cloud Foundry, you must create a statically defined network to host the component virtual machines that constitute the On-Demand Services SDK infrastructure.

Pivotal Cloud Foundry components, like the Cloud Controller and UAA, run on this infrastructure network. On-demand Pivotal Cloud Foundry services may require that you host them on a network that runs separately from this network. You can also deploy tiles on separate service networks to meet your own security requirement.

Pivotal Cloud Foundry v2.0 and Earlier

In Pivotal Cloud Foundry v2.0 and earlier, cloud operators pre-provision service instances from Ops Manager. For each service, Ops Manager allocates and recovers static IP addresses from a pre-defined block of addresses.

To enable on-demand services in Pivotal Cloud Foundry v2.0 and earlier, operators must create a service networks in BOSH Director and select the **Service Network** checkbox. Operators then can select the service network to host on-demand service instances when they configure the tile for that service.

Pivotal Cloud Foundry v2.1 and Later

Pivotal Cloud Foundry v2.1 and later include dynamic networking. In Pivotal Cloud Foundryv2.1 and later, operators can use dynamic networking with asynchronous service provisioning to define dynamically-provisioned service networks. For more information, see Default Network and Service Network.

In Pivotal Cloud Foundry v2.1 and later, on-demand services are enabled by default on all networks. Operators can create separate networks to host services in BOSH Director, but doing so is optional. Operators select which network hosts on-demand service instances when they configure the tile for that service.

Default Network and Service Network

On-demand services use BOSH to dynamically deploy VMs and create single-tenant service instances in a dedicated network. On-demand services use the dynamically-provisioned service network to host single-tenant worker VMs. These worker VMs run as service instances within development spaces.

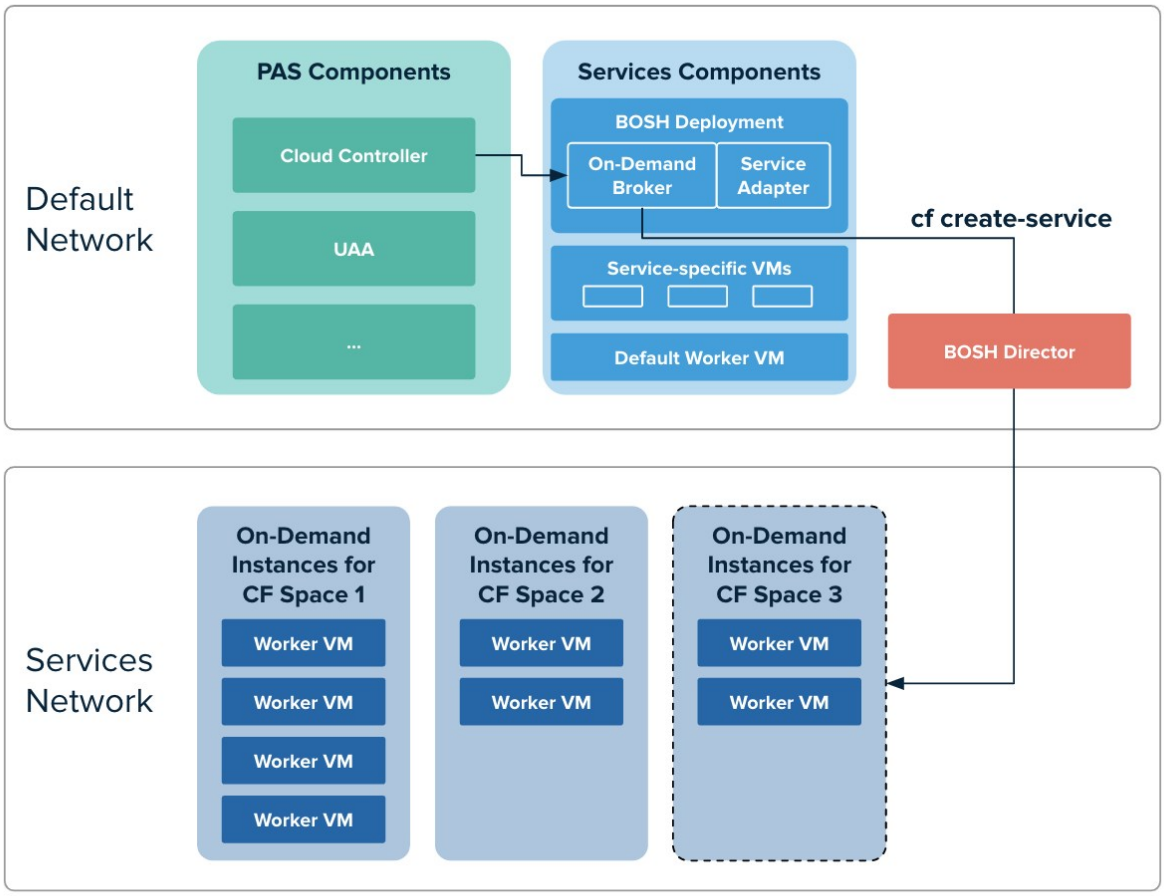
This on-demand architecture has the following advantages:

- Developers can provision IaaS resources for their services instances when the instances are created. This removes the need for operators to pre-provision a fixed amount of IaaS resources when they deploy the service broker.
- Service instances run on a dedicated VM and do not share VMs with unrelated processes. This removes the “noisy neighbor” problem, where an app monopolizes resources on a shared cluster.
- Single-tenant services can support regulatory compliances where sensitive data must be

separated across different machines.

An on-demand service separates operations between the default network and the service network. Shared service components, such as executive controllers and databases, Cloud Controller, UAA, and other on-demand components, run on the default network. Worker pools deployed to specific spaces run on the service network.

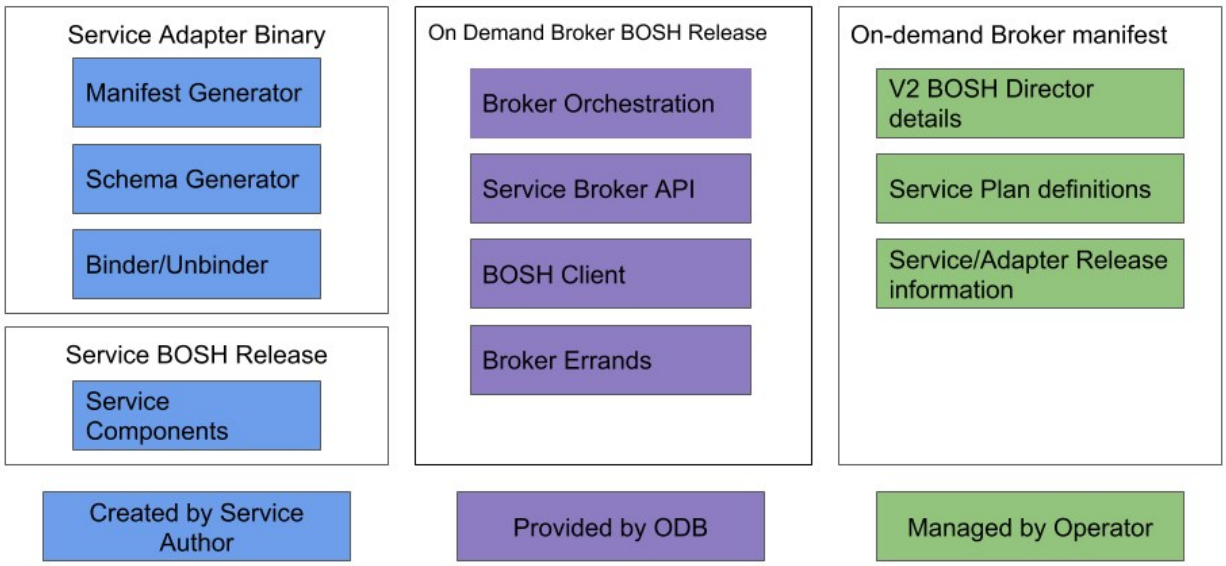
The diagram below shows worker VMs in an on-demand service instance running on a separate services network, while other components run on the default network.



View a larger version of this image

Service Adapters

A service adapter is a binary that is called by the ODB for service-specific tasks. The diagram below shows where responsibility lies for each aspect of the ODB workflow.



The service author can focus on building the BOSH release of their service and provide a service adapter binary that manages manifest generation, schema generation, binding, and unbinding. The ODB manages all interactions with Cloud Foundry and BOSH.

Thanks to BOSH v2, service authors can define resources globally (in **Cloud Config**). This makes manifests portable across BOSH Cloud Provider Interfaces (CPIs) and lets them be substantially smaller than old-style manifests. The ODB takes advantage of other BOSH v2 features as well, including dynamic IP management, availability zones, and links through which deployed BOSH instances can access IP addresses and other information from other instances.

Once an on-demand tile is authored and distributed, the operator installs and configures it the same way they do with any other Pivotal products. In the process, they select which of the tile' s available service plans to offer their developers.

Get Started Using ODB

This documentation provides information for operators and developers in the following sections.

Operator Guide

This section provides information about how to operate an on-demand broker for BOSH operators and Ops Manager operators.

Service and Tile Developer Guide

This section provides information about creating on-demand services and tiles.

Create a pull request or raise an issue on the source for this page in GitHub

Operator Guide

Operating an On-Demand Broker

This topic provides information about operating the on-demand broker for Pivotal Cloud Foundry (PCF) Ops Manager operators and BOSH operators.

Operator Responsibilities

Operators are responsible for the following:

- Requesting appropriate networking rules for on-demand service tiles. See Set Up Networking below.
- Configuring the BOSH Director. See Configure Your BOSH Director below.
- Uploading the required releases for the broker deployment and service instance deployments. See Upload Required Releases below.
- Writing a broker manifest. See Write a Broker Manifest below.
- Managing brokers and service plans. See Broker and Service Management.

Note: Pivotal recommends that you provide documentation when you make changes to the manifest to inform other operators about the new configurations.

Set Up Networking

Before deploying a service tile that uses the on-demand service broker (ODB), you must create networking rules to enable components to communicate with ODB. For instructions for creating networking rules, see the documentation for your IaaS.

The following table lists key components and their responsibilities in the on-demand architecture.

Key Components	Component Responsibilities
BOSH Director	Creates and updates service instances as instructed by ODB.
BOSH Agent	Adds an agent on every VM that it deploys. The agent listens for instructions from the BOSH Director and executes those instructions. The agent receives job specifications from the BOSH Director and uses them to assign a role or job to the VM.
BOSH UAA	Issues OAuth2 tokens for clients to use when they act on behalf of BOSH users.

Pivotal Application Service	Contains the apps that consume services.
ODB	Instructs BOSH to create and update services. Connects to services to create bindings.
Deployed service instance	Runs the given service. For example, a deployed On-Demand Services SDK service instance runs the Redis service.

Regardless of the specific network layout, you must ensure network rules are set up so that connections are open as described in the table below.

This component…	Must communicate with…	Default TCP Port	Communication directions	Notes
ODB	• **BOSH Director** • **BOSH UAA**	• 25555 • 8443	One-way	The default ports are not configurable.
ODB	**Deployed service instances**	Specific to the service (such as RabbitMQ for PCF). May be one or more ports.	One-way	This connection is for administrative tasks. Avoid opening general use, app-specific ports for this connection.
ODB	**PAS**	8443	One-way	The default port is not configurable.
Errand VMs	• **PAS** • **ODB** • **DeployedServiceInstances**	• 8443 • 8080 • Specific to the service. May be one or more ports.	One-way	The default port is not configurable.

BOSH Agent	**BOSH Director**	4222	Two-way	The BOSH Agent runs on every VM in the system, including the BOSH Director VM. The BOSH Agent initiates the connection with the BOSH Director. The default port is not configurable.
Deployed apps on PAS	**Deployed service instances**	Specific to the service. May be one or more ports.	One-way	This connection is for general use, app-specific tasks. Avoid opening administrative ports for this connection.
PAS	**ODB**	8080	One-way	This port can be different for individual services. This port can also be configurable by the operator if allowed by the tile developer.

Configure Your BOSH Director

See the following topics for how to set up your BOSH Director:

- Software Requirements
- Configure CA Certificates for TLS Communication
- BOSH Teams
- Cloud Controller

Software Requirements

The On-Demand Broker requires the following:

- BOSH Director v266.12.0 or v267.6.0 and later. To install the BOSH Director, see Quick Start in the BOSH documentation.
- cf-release v1.10.0 or later (PCF v2.0 or later).

Notes:

- ODB does not support BOSH Windows.
- Service instance lifecycle errands require BOSH Director v261 on PCF v1.10 or later. For more information, see Service Instance Lifecycle Errands below.

Configure CA Certificates for TLS Communication

There are two kinds of communication in ODB that use transport layer security (TLS) and need to validate certificates using a certificate authority (CA) certificate:

- ODB to BOSH Director
- ODB to Cloud Foundry API (Cloud Controller)

The CA certificates used to sign the BOSH and Cloud Controller certificates are often generated by BOSH, CredHub, or a customer security team, and so are not publicly trusted certificates. This means Pivotal might need to provide the CA certificates to ODB to perform the required validation.

ODB to BOSH Director

In some rare cases where the BOSH Director is not installed through Ops Manager, BOSH can be configured to be publicly accessible with a domain name and a TLS certificate issued by a public certificate authority. In such a case, you can navigate to https://BOSH-DOMAIN-NAME:25555/info in a browser and see a trusted certificate padlock in the browser address bar.

ODB can then be configured to use this address for BOSH and will not require a CA certificate to be provided. The public CA certificate would already be present on the ODB VM.

By contrast, BOSH is usually only accessible on an internal network. It uses a certificate signed by an internal CA. The CA certificate must be provided in the broker configuration so that ODB can validate the BOSH Director's certificate. ODB always validates BOSH TLS certificates.

You have two options for providing a CA certificate to ODB for validation of the BOSH certificate. You can add the BOSH Director's root certificate to the ODB manifest or you can use BOSH's `trusted_certs` feature to add a self-signed CA certificate to each VM that BOSH deploys.

- To add the BOSH Director's root certificate to the ODB manifest, edit the manifest as below:

  ```
  bosh:
    root_ca_cert: ROOT-CA-CERT
  ```

  Where `ROOT-CA-CERT` is the root certificate authority (CA) certificate. This is the certificate used when following the steps in Configuring SSL Certificates in the BOSH documentation.

  For example:

  ```
  Instance_groups:
      - Name: broker
        Jobs:
          - Name: broker
            Properties:
              bosh:
                root_ca_cert:
                  -----BEGIN CERTIFICATE-----
                  EXAMPLExxOFxxAxxCERTIFICATE
                  ...
                  -----END CERTIFICATE-----
                authentication:
                ...
  ```

- To use BOSH's `trusted_certs` feature to add a self-signed CA certificate to each VM that BOSH deploys, follow the steps below.

  1. Generate and use self-signed certificates for the BOSH Director and User Account and Authentication (UAA) through the `trusted_certs` feature. For instructions, see Configuring Trusted Certificates in the BOSH documentation.

  2. Add trusted certificates to your BOSH Director. For instructions, see Configuring SSL Certificates in the BOSH documentation.

ODB to Cloud Controller

Optionally, you can configure a separate root CA certificate that is used when ODB communicates with the Cloud Foundry API (Cloud Controller). This is necessary if the Cloud Controller is configured with a certificate not trusted by the broker.

For an example of how to add a separate root CA certificate to the manifest, see the line containing `CA-CERT-FOR-CLOUD-CONTROLLER` in the manifest snippet in Starter Snippet for Your Broker below.

Use BOSH Teams

You can use BOSH teams to further control how BOSH operations are available to different clients. For more information about BOSH teams, see Using BOSH Teams in the BOSH documentation.

To use BOSH teams to ensure that your on-demand service broker client can only modify deployments it created, do the following:

1. Run the following UAA CLI (UAAC) command to create the client:

   ```
   uaac client add CLIENT-ID \
     --secret CLIENT-SECRET \
     --authorized_grant_types "refresh_token password client_credentials" \
     --authorities "bosh.teams.TEAM-NAME.admin"
   ```

   Where:

   - `CLIENT-ID` is your client ID.
   - `CLIENT-SECRET` is your client secret.
   - `TEAM-NAME` is the name of the team authorized to modify this deployment.

   For example:

   ```
   uaac client add admin \
     --secret 12345679 \
     --authorized_grant_types "refresh_token password client_credentials" \
     --authorities "bosh.teams.my-team.admin"
   ```

   For more information about using the UAAC, see Creating and Managing Users with the UAA CLI (UAAC).

2. Configure the broker's BOSH authentication.

   For example:

   ```
   instance_groups:
     - name: broker
       ...
       jobs:
         - name: broker
           ...
           properties:
             ...
             bosh:
               url: DIRECTOR-URL
               root_ca_cert: CA-CERT-FOR-BOSH-DIRECTOR # optional, see SSL certifi
   ```

```
cates
            authentication:
              uaa:
                client_id: BOSH-CLIENT-ID
                client_secret: BOSH-CLIENT-SECRET
```

Where the `BOSH-CLIENT-ID` and `BOSH-CLIENT-SECRET` are the `CLIENT-ID` and `CLIENT-SECRET` you provided in step 1.

The broker can then only perform BOSH operations on deployments it has created. For a more detailed manifest snippet, see Starter Snippet for Your Broker below.

For more information about securing how ODB uses BOSH, see Security.

Set Up Cloud Controller

ODB uses the Cloud Controller as a source of truth for service offerings, plans, and instances.

To reach the Cloud Controller, configure ODB with either client or user credentials in the broker manifest. For more information, see Write a Broker Manifest below.

Note: The client or user must have the following permissions.

- **If using client credentials** then, as of Cloud Foundry v238, the UAA client must have the authority `cloud_controller.admin`.
- **If using user credentials** then the user must be a member of the `scim.read` and `cloud_controller.admin` groups.

The following is an example broker manifest snippet for the client credentials:

```
authentication:
  ...
  client_credentials:
    client_id: UAA-CLIENT-ID
    secret: UAA-CLIENT-SECRET
```

The following is an example broker manifest snippet for the user credentials:

```
authentication:
  ...
  user_credentials:
    username: CF-ADMIN-USERNAME
    password: CF-ADMIN-PASSWORD
```

Upload Required Releases

Upload the following releases to your BOSH Director:

- **On Demand Service Broker (ODB)**—Download ODB from Pivotal Network.
- **Your service adapter**—Get the service adapter from the release author.
- **Your service release**—Get the service release from the release author.

- **BOSH Process Manager (BPM) release**—Get the BPM release from BOSH releases in GitHub. You might not need to do this if the BPM release is already uploaded.

To upload a release to your BOSH Director, do the following:

1. Run the following command.

   ```
   bosh -e BOSH-DIRECTOR-NAME upload-release RELEASE-FILE-NAME.tgz
   ```

   Example command for ODB:

   ```
   $ bosh -e lite upload-release on-demand-service-broker-0.22.0.tgz
   ```

   Example commands for service adapter or service release:

   ```
   $ bosh -e lite upload-release my-service-release.tgz
   ```

   ```
   $ bosh -e lite upload-release my-service-adapter.tgz
   ```

Write a Broker Manifest

There are two parts to writing your broker manifest. You must:

- Configure Your Broker
- Configure Your Service Catalog and Plan Composition

If you are unfamiliar with writing BOSH v2 manifests, see Deployment Config.

For example manifests, see the following:

- For a Redis service—redis-example-service-adapter-release in GitHub.
- For a Kafka service—kafka-example-service-adapter-release in GitHub.

Configure Your Broker

Your manifest must contain exactly one non-errand instance group that is co-located with both of the following:

- The broker job from `on-demand-service-broker`
- Your service adapter job from your service adapter release

The broker is stateless and does not need a persistent disk. Its VM type can be small: a single CPU and 1 GB of memory is sufficient in most cases.

Starter Snippet for Your Broker

Use the snippet below to help you to configure your broker. The snippet uses BOSH v2 syntax as well as global cloud config and job-level properties.

For examples of complete broker manifests, see Write a Broker Manifest above.

WARNING: The `disable_ssl_cert_verification` option is dangerous and should be set to `false` in production.

```
addons:
  # Broker uses BPM to isolate co-located BOSH jobs from one another
  - name: bpm
    jobs:
    - name: bpm
      release: bpm
instance_groups:
  - name: NAME-OF-YOUR-CHOICE
    instances: 1
    vm_type: VM-TYPE
    stemcell: STEMCELL
    networks:
      - name: NETWORK
    jobs:
      - name: SERVICE-ADAPTER-JOB-NAME
        release: SERVICE-ADAPTER-RELEASE
      - name: broker
        release: on-demand-service-broker
        properties:
          # choose a port and basic authentication credentials for the broker:
          port: BROKER-PORT
          username: BROKER-USERNAME
          password: BROKER-PASSWORD
          # optional - defaults to false. This should not be set to true in production
.
          disable_ssl_cert_verification: TRUE|FALSE
          # optional - defaults to 60 seconds. This enables the broker to gracefully w
ait for any open requests to complete before shutting down.
          shutdown_timeout_in_seconds: 60
          # optional - defaults to false. This enables BOSH operational errors to be d
isplayed for the CF user.
          expose_operational_errors: TRUE|FALSE
          # optional - defaults to false. If set to true, plan schemas are included in
 the catalog, and the broker fails if the adapter does not implement generate-plan-sch
emas.
          enable_plan_schemas: TRUE|FALSE
          cf:
            url: CF-API-URL
            # optional - see the Configure CA Certificates section above:
            root_ca_cert: CA-CERT-FOR-CLOUD-CONTROLLER
            # either client_credentials or user_credentials, not both as shown:
            authentication:
              url: CF-UAA-URL
              client_credentials:
                # with cloud_controller.admin authority and client_credentials in the
authorized_grant_type:
                client_id: UAA-CLIENT-ID
                secret: UAA-CLIENT-SECRET
              user_credentials:
                # in the cloud_controller.admin and scim.read groups:
                username: CF-ADMIN-USERNAME
                password: CF-ADMIN-PASSWORD
          bosh:
            url: DIRECTOR-URL
            # optional - see the Configure CA Certificates section above:
```

```
            root_ca_cert: CA-CERT-FOR-BOSH-DIRECTOR
            # either basic or uaa, not both as shown, see
            authentication:
              basic:
                username: BOSH-USERNAME
                password: BOSH-PASSWORD
              uaa:
                client_id: BOSH-CLIENT-ID
                client_secret: BOSH-CLIENT-SECRET
          service_adapter:
            # optional - provided by the service author. Defaults to /var/vcap/package
s/odb-service-adapter/bin/service-adapter.
            path: PATH-TO-SERVICE-ADAPTER-BINARY
            # optional - Filesystem paths to be mounted for use by the service adapter
. These should include the paths to any config files.
            mount_paths: [ PATH-TO-SERVICE-ADAPTER-CONFIG ]
          # There are more broker properties that are discussed below
```

Configure Your Service Catalog and Plan Composition

Use the following sections as a guide to configure the service catalog and compose plans in the properties section of broker job. For an example snippet, see the Starter Snippet for the Service Catalog and Plans below.

Configure the Service Catalog

When configuring the service catalog, supply the following:

- **The release jobs specified by the service author:**
  - Supply each release job exactly once.
  - You can include releases that provide many jobs, as long as each required job is provided by exactly one release.

- **Stemcells:**

  Note: If you are using Xenial stemcells, you must update any BOSH add-ons to support Xenial stemcells. For links to instructional topics about updating see Update Add-ons to Run with Xenial Stemcell.

  - These are used on each VM in the service deployments.
  - Use exact versions for releases and stemcells. The use of `latest` and floating stemcells are not supported.

- **Cloud Foundry service metadata for the service offering:**
  - This metadata is aggregated in the Marketplace and displayed in Apps Manager and the cf CLI.
  - You can use other arbitrary field names as needed in addition to the Open Service Broker API (OSBAPI) recommended fields. For information about the recommended fields for service metadata, see the Open Service Broker API Profile.

Compose Plans

Service authors do not define plans, but instead expose plan properties. Operators compose plans consisting of combinations of these properties, along with IaaS resources and catalog metadata.

When composing plans, supply the following:

- **Cloud Foundry plan metadata for each plan:**

  You can use other arbitrary field names in addition to the OSBAPI recommended fields. For information about the recommended fields for plan metadata, see the Open Service Broker API Profile in GitHub.

- **Resource mapping:**
  - For each plan, supply resource mapping for each instance group that service authors specify.
  - The resource values must correspond to valid resource definitions in the BOSH Director's global cloud config.
  - Service authors might recommend resource configuration. For example, in single-node Redis deployments, an instance count greater than one does not make sense. Here, you can configure the deployment to span multiple availability zones (AZs). For how to do this, see Availability Zones in the BOSH documentation.
  - Service authors might provide errands for the service release. You can add an instance group of type `errand` by setting the `lifecycle` field. For an example, see `register-broker` in the kafka-example-service-adapter-release in GitHub.

- **Values for plan properties:**
  - Plan properties are key-value pairs defined by the service authors. For example, including a boolean to enable disk persistence for Redis or a list of strings representing RabbitMQ plugins to load.
  - The service author should document whether a plan property:
    - Is mandatory or optional
    - Precludes the use of another
    - Affects recommended instance group to resource mappings
  - You can also specify global properties at the service offering level, where they are applied to every plan. If there is a conflict between global and plan-level properties, the plan properties take precedence.

- **(Optional) Provide an update block for each plan**
  - You might require plan-specific configuration for BOSH's update instance operation. ODB passes the plan-specific update block to the service adapter.
  - Plan-specific update blocks should have the same structure as the update block in a

BOSH manifest. See Update Block in the BOSH documentation.

- The service author can define a default update block to be used when a plan-specific update block is not provided, if the service adapter supports configuring update blocks in the manifest.

- **(Optional) Maintenance Information**
  - Maintenance information is used to uniquely describe the deployed version of a service instance. It is made up of public and private key/value pairs defined at the catalog and plan levels. It is displayed in the service catalog at plan level with plan-level maintenance information aggregated into the catalog-level maintenance information. The public information is exposed in the service catalog, while all the private information is aggregated and hashed into a value representing those private values.
  - Maintenance information which is common to all plans should be defined at the service catalog level.
  - Plan-specific maintenance information should be defined at the plan level. Where a key is redefined at the plan level, it overrides the service catalog level value.
  - Pivotal recommends using YAML anchors and references to avoid repeating maintenance information values within the manifest. For instance, the stemcell version can be anchors with the `&stemcellVersion` annotation, and then referenced in the maintenance information with the `*stemcellVersion` tag.

Starter Snippet for the Service Catalog and Plans

Append the snippet below to the properties section of the broker job that you configured in *Configure Your Broker*. Ensure that you provide the required information listed in Configure Your Service Catalog and Plan Composition above.

For examples of complete broker manifests, see Write a Broker Manifest above.

```
service_deployment:
  releases:
    - name: SERVICE-RELEASE
      # exact release version:
      version: SERVICE-RELEASE-VERSION
      # service author specifies the list of jobs required:
      jobs: [RELEASE-JOBS-NEEDED-FOR-DEPLOYMENT-AND-LIFECYCLE-ERRANDS]
  # every instance group in the service deployment has the same stemcell:
  stemcells:
    - os: SERVICE-STEMCELL
      # exact stemcell version:
      version: &stemcellVersion SERVICE-STEMCELL-VERSION
service_catalog:
  id: CF-MARKETPLACE-ID
  service_name: CF-MARKETPLACE-SERVICE-OFFERING-NAME
  service_description: CF-MARKETPLACE-DESCRIPTION
  bindable: TRUE|FALSE
  # optional:
  plan_updatable: TRUE|FALSE
  # optional:
  tags: [TAGS]
```

```
  # optional:
  requires: [REQUIRED-PERMISSIONS]
  # optional:
  dashboard_client:
    id: DASHBOARD-OAUTH-CLIENT-ID
    secret: DASHBOARD-OAUTH-CLIENT-SECRET
    redirect_uri: DASHBOARD-OAUTH-REDIRECT-URI
  # optional:
  metadata:
    display_name: DISPLAY-NAME
    image_url: IMAGE-URL
    long_description: LONG-DESCRIPTION
    provider_display_name: PROVIDER-DISPLAY-NAME
    documentation_url: DOCUMENTATION-URL
    support_url: SUPPORT-URL
  # optional - applied to every plan:
  global_properties: {}
  global_quotas: # optional
    # the maximum number of service instances across all plans:
    service_instance_limit: INSTANCE-LIMIT
    # optional - resource usage limits, determined by the 'cost' of each service insta
nce plan:
    resource_limits:
      ips: RESOURCE-LIMIT
      memory: RESOURCE-LIMIT
  # optional - applied to every plan.
  maintenance_info:
    # keys under public are visible in service catalog
    public:
      # reference to stemcellVersion anchor above
      stemcell_version: *stemcellVersion
      # arbitrary public maintenance_info
      kubernetes_version: 1.13
      # arbitrary public maintenance_info
      docker_version: 18.06.1
     # all keys under private are hashed to single SHA value in service catalog
    private:
      # example of private data that would require a service update to change
      log_aggregrator_mtls_cert: *YAML_ANCHOR_TO_MTLS_CERT
  plans:
    - name: CF-MARKETPLACE-PLAN-NAME
      # optional - used by the cf CLI to display whether this plan is "free" or "paid"
:
      free: TRUE|FALSE
      plan_id: CF-MARKETPLACE-PLAN-ID
      description: CF-MARKETPLACE-DESCRIPTION
      # optional - enable by default.
      cf_service_access: ENABLE|DISABLE|MANUAL
      # optional -  if specified, this takes precedence over the bindable attribute of
 the service:
      bindable: TRUE|FALSE
      # optional:
      metadata:
        display_name: DISPLAY-NAME
        bullets: [BULLET1, BULLET2]
        costs:
          - amount:
              CURRENCY-CODE-STRING: CURRENCY-AMOUNT-FLOAT
            unit: FREQUENCY-OF-COST
```

```
# optional – the 'cost' of each instance in terms of resource quotas:
resource_costs:
  memory: AMOUNT-OF-RESOURCE-IN-THIS-PLAN
# optional:
quotas:
  # the maximum number of service instances for this plan:
  service_instance_limit: INSTANCE-LIMIT
  # optional - resource usage limits for this plan:
  resource_limits:
    memory: RESOURCE-LIMIT
# resource mapping for the instance groups defined by the service author:
instance_groups:
  - name: SERVICE-AUTHOR-PROVIDED-INSTANCE-GROUP-NAME
    vm_type: VM-TYPE
    # optional:
    vm_extensions: [VM-EXTENSIONS]
    instances: &instanceCount INSTANCE-COUNT
    networks: [NETWORK]
    azs: [AZ]
    # optional:
    persistent_disk_type: DISK
    # optional:
  - name: SERVICE-AUTHOR-PROVIDED-LIFECYCLE-ERRAND-NAME
    lifecycle: errand
    vm_type: VM-TYPE
    instances: INSTANCE-COUNT
    networks: [NETWORK]
    azs: [AZ]
# valid property key-value pairs are defined by the service author:
properties: {}
# optional
maintenance_info:
  # optional - keys merge with catalog level public maintenance_info keys
  public:
    # refers to anchor in instance group above
    instance_count: *instanceCount
  # optional
  private: {}
# optional:
update:
  # optional:
  canaries: 1
  # required:
  max_in_flight: 2
  # required:
  canary_watch_time: 1000-30000
  # required:
  update_watch_time: 1000-30000
  # optional:
  serial: true
# optional:
lifecycle_errands:
  # optional:
  post_deploy:
    - name: ERRAND-NAME
      # optional - for co-locating errand:
      instances: [INSTANCE-NAME, ...]
    - name: ANOTHER_ERRAND_NAME
  # optional:
```

```
    pre_delete:
      - name: ERRAND-NAME
        # optional - for co-locating errand:
        instances: [INSTANCE-NAME, ...]
```

(Optional) Access Manifest Secrets at Bind Time

Note: This feature does not work if you have configured `use_stdin` to be false.

A service adapter might need to access secrets embedded in a service instance manifest when processing a create binding request. For example, it might need credentials with sufficient privileges to add a new user to a service instance. These credentials are in the service instance manifest. ODB passes this manifest to the adapter in the `create-binding` call.

Secrets in the manifest can be:

- BOSH variables
- Literal BOSH CredHub references
- Plain text

If you use BOSH variables or literal CredHub references in your manifest, do the following in the ODB manifest so that the service adapter can access the secrets:

1. Set the `enable_secure_manifests` flag to `true`.

   For example:

   ```
   instance_groups:
     - name: broker
       ...
       jobs:
         - name: broker
           ...
           properties:
             ...
             enable_secure_manifests: true
             ...
   ```

2. Supply details for accessing the credentials stored in BOSH CredHub. Replace the placeholder text below with your values for accessing CredHub:

   ```
   instance_groups:
     - name: broker
       ...
       jobs:
         - name: broker
           ...
           properties:
             ...
             enable_secure_manifests: true
             bosh_credhub_api:
               url: https://BOSH-CREDHUB-ADDRESS:8844/
               root_ca_cert: BOSH-CREDHUB-CA-CERT
   ```

```
            authentication:
              uaa:
                client_credentials:
                  client_id: BOSH-CREDHUB-CLIENT-ID
                  client_secret: BOSH-CREDHUB-CLIENT-SECRET
```

When the `enable_secure_manifests` flag is set to `true`, ODB queries BOSH and its CredHub instance for secret values. ODB then generates a map of all manifest variable names and CredHub references to secret values in the manifest. ODB passes this map to the service adapter's `create-binding` call. For an example of the JSON in the `create-binding` call, see the Service Adapter Interface Reference documentation.

If ODB cannot resolve a particular secret, it logs the failure and omits the unresolved secret from the passed secrets map. It is the responsibility of the adapter to handle missing secrets based on whether they are required for binding creation.

Note: ODB does not fail if it cannot resolve a secret.

(Optional) Enable Secure Binding

Note: This feature does not work if you have configured `use_stdin` to be false.

If you enable secure binding, binding credentials are stored securely in runtime CredHub. When users create bindings or service keys, ODB passes a secure reference to the service credentials through the network instead of in plaintext.

Requirements

To store service credentials in runtime CredHub, your deployment must meet the following requirements:

- It must be able to connect to runtime CredHub v1.6.x or later. This might be provided as part of your Cloud Foundry deployment.
- Your instance group must have access to the local DNS provider. This is because the address for runtime CredHub is a local domain name.

Note: Pivotal recommends using BOSH DNS as a DNS provider. If you are using PCF v2.4 or later, you cannot use consul as a DNS provider because consul server VMs have been removed in Pivotal Application Service (PAS) v2.4.

Procedure for Enabling Secure Binding

To enable secure binding, do the following:

1. Set up a new runtime CredHub client in Cloud Foundry UAA with `credhub.write` and `credhub.read` in its list of scopes. For how to do this, see Creating and Managing Users with the UAA CLI (UAAC) in the Cloud Foundry documentation.

2. Update the broker job in the ODB manifest to consume the runtime CredHub link.

   For example:

   ```
   instance_groups:
     - name: broker
       ...
       jobs:
         - name: broker
           consumes:
             credhub:
               from: credhub
               deployment: cf
   ```

3. Update the broker job in the ODB manifest to include the `secure_binding_credentials` section. The CA certificate can be a reference to the certificate in the cf deployment or inserted manually.

   For example:

   ```
   instance_groups:
     - name: broker
       ...
       jobs:
         - name: broker
           ...
           properties:
             ...
             secure_binding_credentials:
               enabled: true
               authentication:
                 uaa:
                   client_id: NEW-CREDHUB-CLIENT-ID
                   client_secret: NEW-CREDHUB-CLIENT-SECRET
                   ca_cert: ((cf.uaa.ca_cert))
   ```

   Where `NEW-CREDHUB-CLIENT-ID` and `NEW-CREDHUB-CLIENT-SECRET` are the runtime CredHub client credentials you created in step 1.

For a more detailed manifest snippet, see Starter Snippet for Your Broker above.

How Credentials Are Stored on Runtime CredHub

The credentials for a given service binding are stored with the following format:

```
/C/:SERVICE-GUID/:SERVICE-INSTANCE-GUID/:BINDING-GUID/CREDENTIALS
```

The plaintext credentials are stored in runtime CredHub under this key, and the key is available under the `VCAP_SERVICES` environment variable for the app.

(Optional) Enable Plan Schemas

As of OSBAPI Spec v2.13 ODB supports enabling plan schemas. For more information, see OSBAPI

Spec v2.13 in GitHub.

When this feature is enabled, the broker validates incoming configuration parameters against a schema during the provision, binding, and update of service instances. The broker produces an error if the parameters do not conform.

To enable plan schemas, do the following:

1. Ensure that the service adapter implements the command `generate-plan-schemas`. When it is not implemented, the broker fails to deploy. For more information about this command, see generate-plan-schemas.
2. In the manifest, set the `enable_plan_schemas` flag to `true` as shown below. The default is `false`.

   ```
   instance_groups:
     - name: broker
       ...
       jobs:
         - name: broker
           ...
           properties:
             ...
             enable_plan_schemas: true
   ```

   For a more detailed manifest snippet, see Starter Snippet for Your Broker above.

(Optional) Register the Route to the Broker

You can register a route to the broker using the `route_registrar` job from the routing release. The `route_registrar` job achieves the following:

- Load balances multiple instances of ODB using the Cloud Foundry router
- Allows access to ODB from the public internet

For more information, see route_registrar job.

To register the route, co-locate the `route_registrar` job with `on-demand-service-broker`:

1. Download the routing release. See cf-routing Release for more information about doing so.
2. Upload the routing release to your BOSH Director.
3. Add the `route_registrar` job to your deployment manifest and configure it with an HTTP route. This creates a URI for your broker.

   Note: You must use the same port for the broker and the route. The broker defaults to 8080.

   For how to configure the `route_registrar` job, see routing release in GitHub.
4. If you configure a route, set the `broker_uri` property in the register-broker errand.

(Optional) Set Service Instance Quotas

Set service instance quotas to limit the number of service instances ODB can create. You can set these quotas for service instances:

- **Global quotas** – limit the number of service instances across all plans
- **Plan quotas** – limit the number of service instances for a given plan

Note: These limits do not include orphaned deployments. See List Orphan Deployments and Delete Orphaned Deployments for more information.

When creating a service instance, ODB checks the global service instance limit. If this limit has not been reached, ODB checks the plan service instance limit.

Procedure for Setting Service Instance Quotas

To set service instance quotas, do the following in the manifest:

1. Add a quotas section for the type of quota you want to use.

    - **For global quotas** add `global_quotas` in the service catalog, as in the example below:

      ```
      service_catalog:
        ...
        global_quotas:
          service_instance_limit: INSTANCE-LIMIT
          ...
      ```

    - **For plan quotas** add `quotas` to the plans you want to limit, as in the example below:

      ```
      service_catalog:
        ...
        plans:
          - name: CF-MARKETPLACE-PLAN-NAME
            quotas:
              service_instance_limit: INSTANCE-LIMIT
      ```

      Where `INSTANCE-LIMIT` is the maximum number of service instances allowed.

For a more detailed manifest snippet, see the Starter Snippet for the Service Catalog and Plans above.

(Optional) Set Resource Quotas

Set resource quotas to limit resources, such as memory or disk, more effectively when combining plans that consume different amounts of resources. You can set these quotas for service resources:

- **Global resource quotas** – limit how much of a certain resource is consumed across all plans. ODB allows new instances to be created until their total resources reach the global quota.
- **Plan resource quotas** – limit how much of a certain resource is consumed by a specific plan.

Note: These limits do not include orphaned deployments. See List Orphan Deployments and Delete Orphaned Deployments for more information.

When creating a service instance, ODB checks the global resource limit. If this limit has not been reached, ODB checks the plan resource limit.

Procedure for Setting Service Resource Quotas

To set resource quotas, do the following in the manifest:

1. Add a quotas section for the type of quota you want to use by entering the following.

   ```
   quotas:
     resource_limits:
       RESOURCE-NAME: RESOURCE-LIMIT
   ```

   Where:

   - `RESOURCE-NAME` is a string defining the resource you want to limit.
   - `RESOURCE-LIMIT` is a value for the maximum allowed for each resource.

   For example:

   - **For global quotas** add `global_quotas` in the service catalog, as in this example:

     ```
     service_catalog:
       ...
       global_quotas:
         resource_limits:
           ips: 50
           memory: 150
     ```

   - **For plan quotas** add `quotas` in the plans you want to limit, as in this example:

     ```
     service_catalog:
       ...
       plans:
         - name: my-plan
           quotas:
             resource_limits:
               memory: 25
     ```

2. Add `resource_costs` in each plan to define the amount of resources your plan allocates to each service instance. The key is string-matched against keys in the global- and plan-level resource quotas. See the example below.

   ```
   resource_costs:
     RESOURCE-NAME: AMOUNT-OF-RESOURCE
   ```

   Where:

   - `RESOURCE-NAME` is a string defining the resource you want to limit.

- `AMOUNT-OF-RESOURCE` is the amount of the resource allocated to each service instance of this plan.

For example:

```
service_catalog:
  ...
  plans:
    - name: my-plan
      resource_costs:
        memory: 5
```

For a more detailed manifest snippet, see the Starter Snippet for the Service Catalog and Plans above.

(Optional) Configure Service Metrics

The ODB BOSH release contains a metrics job that can be used to emit metrics when co-located with the Pivotal Cloud Foundry Service Metrics SDK. To do this, you must include the Loggregator release.

To download the Pivotal Cloud Foundry Service Metrics SDK, see Pivotal Network.

Add the following jobs to the broker instance group:

```
- name: service-metrics
  release: service-metrics
  properties:
    service_metrics:
      execution_interval_seconds: INTERVAL-BETWEEN-SUCCESSIVE-METRICS-COLLECTIONS
      origin: ORIGIN-TAG-FOR-METRICS
      monit_dependencies: [broker]
      ....snip....
      LOGGREGATOR-CONFIGURATION
      ....snip....
- name: service-metrics-adapter
  release: ODB-RELEASE
```

Where:

- `INTERVAL-BETWEEN-SUCCESSIVE-METRICS-COLLECTIONS` is the interval in seconds between successive metrics collections.
- `ORIGIN-TAG-FOR-METRICS` is the origin tag for metrics.
- `LOGGREGATOR-CONFIGURATION` is your Loggregator configuration. For example manifests, see service-metrics-release in GitHub.
- `ODB-RELEASE` is the on-demand broker release.

For an example of how the service metrics can be configured for an on-demand-broker deployment, see the kafka-example-service-adapter-release manifest in GitHub.

Pivotal has tested this example configuration with Loggregator v58 and service-metrics v1.5.0.

For more information about service metrics, see Service Metrics for Pivotal Cloud Foundry.

(Optional) Obtain BOSH DNS Addresses for Binding Creation and Deletion

You can configure ODB to retrieve BOSH DNS addresses for service instances. These addresses are passed to the service adapter when you create or delete a binding.

Requirements

- A service that has this feature enabled in the service adapter
  For information for service authors about how to enable this feature for their on-demand service, see Enable ODB to Obtain BOSH DNS Addresses.
- BOSH Director v266.12 or v267.6 and later, available in Ops Manager v2.2.5 and later

Procedure

To enable ODB to obtain BOSH DNS addresses for binding creation and deletion, do the following:

1. In the manifest, configure the `binding_with_dns` property on plans that require DNS addresses to create and delete bindings.

   For more information about the properties to add, see Options for *binding_with_dns* below.

   For example:

   ```
   service_catalog:
     ...
     plans:
       ...
       - name: plan-requiring-dns-addresses
         ...
         binding_with_dns:                 # add this section
           - name: leader-address
             link_provider: example-link-1
             instance_group: leader-node
           - name: follower-address
             link_provider: example-link-2
             instance_group: follower-node
             properties:
               azs: [z1, z2]
               status: healthy
   ```

   Each entry in `binding_with_dns` is converted to a BOSH DNS address that is passed to the service adapter when you create a binding.

Options for *binding_with_dns*

The following table provides descriptions of the properties to add to the `binding_with_dns` section:

Property	Description	Mandatory/Optional
`name`	An arbitrary identifier used to identify the address when creating a binding	Mandatory

Property	Description	Mandatory/Optional
`link_provider`	This is the exposed name of the link. You can find this in the documentation for the service and under `provides.name` in the release `spec` file. You can override it in the deployment manifest by setting the `as` property of the link.	Mandatory
`instance_group`	This is the name of the instance group sharing the link. The resultant DNS address resolves to IP addresses of this instance group.	Mandatory
`properties.azs`	This is a list of availability zone names. When this is provided, the resultant DNS address resolves to IP addresses in these zones.	Optional
`properties.status`	This is a filter for link address status (healthy, unhealthy, all, default). When this is provided, the resultant DNS address resolves to IP addresses with this status.	Optional

About Broker Startup Checks

The ODB does the following startup checks:

- It verifies that the CF and BOSH versions satisfy the minimum versions required. If your service offering includes lifecycle errands, the minimum required version for BOSH is higher. For more information, see Configure Your BOSH Director above.

  If your system does not meet minimum requirements, you see an insufficient version error. For example:

  ```
  CF API error: Cloud Foundry API version is insufficient, ODB requires CF v238+.
  ```

- It verifies that, for the service offering, no plan IDs have changed for plans that have existing service instances. If there are instances, you see the following error:

  ```
  You cannot change the plan_id of a plan that has existing service instances.
  ```

About Broker Shutdown

The broker tries to wait for any incomplete HTTPS requests to complete before shutting down. This reduces the risk of leaving orphan deployments in the event that the BOSH Director does not respond to the initial `bosh deploy` request.

You can determine how long the broker waits before being forced to shut down by using the `broker.shutdown_timeout` property in the manifest. The default is 60 seconds. For more information, see Write a Broker Manifest above.

Service Instance Lifecycle Errands

Note: This feature requires BOSH Director v261 or later.

Service instance lifecycle errands allow additional short-lived jobs to run as part of service instance deployment. A deployment is only considered successful if all lifecycle errands exit successfully.

The service adapter must offer the errands as part of the service instance deployment.

ODB supports the following lifecycle errands:

- `post_deploy` runs after creating or updating a service instance. An example use case is running a health check to ensure the service instance is functioning.
  For more information about the workflow, see Create or Update Service Instance with Post-Deploy Errands.

- `pre_delete` runs before the deletion of a service instance. An example use case is cleaning up data before a service shutdown. For more information about the workflow, see Delete a Service Instance with Pre-Delete Errands.

Enable Service Instance Lifecycle Errands

Service instance lifecycle errands are configured on a per-plan basis. Lifecycle errands do not run if you change a plan's lifecycle errand configuration while an existing deployment is in progress.

To enable lifecycle errands, do the following steps.

1. Add each errand job in the following manifest places:
   - `service_deployment`
   - The plan's `lifecycle_errands` configuration
   - The plan's `instance_groups`

   Below is an example manifest snippet that configures lifecycle errands for a plan:

```
service_deployment:
  releases:
    - name: SERVICE-RELEASE
      version: SERVICE-RELEASE-VERSION
      jobs:
      - SERVICE-RELEASE-JOB
      - POST-DEPLOY-ERRAND-JOB
      - PRE-DELETE-ERRAND-JOB
      - ANOTHER-POST-DEPLOY-ERRAND-JOB
service_catalog:
  plans:
    - name: CF-MARKETPLACE-PLAN-NAME
      lifecycle_errands:
        post_deploy:
          - name: POST-DEPLOY-ERRAND-JOB
          - name: ANOTHER-POST-DEPLOY-ERRAND-JOB
            disabled: true
        pre_delete:
          - name: PRE-DELETE-ERRAND-JOB
      instance_groups:
        - name: SERVICE-RELEASE-JOB
          ...
        - name: POST-DEPLOY-ERRAND-JOB
          lifecycle: errand
```

```
          vm_type: VM-TYPE
          instances: INSTANCE-COUNT
          networks: [NETWORK]
          azs: [AZ]
        - name: ANOTHER-POST-DEPLOY-ERRAND-JOB
          lifecycle: errand
          vm_type: VM-TYPE
          instances: INSTANCE-COUNT
          networks: [NETWORK]
          azs: [AZ]
        - name: PRE-DELETE-ERRAND-JOB
          lifecycle: errand
          vm_type: VM-TYPE
          instances: INSTANCE-COUNT
          networks: [NETWORK]
          azs: [AZ]
```

Where `POST-DEPLOY-ERRAND-JOB` is the errand job you want to add.

(Optional) Enable Co-located Errands

Note: This feature requires BOSH Director v263 or later.

You can run both `post-deploy` and `pre-delete` errands as co-located errands. Co-located errands run on an existing service instance group instead of a separate one. This avoids additional resource allocation.

Like other lifecycle errands, co-located errands are deployed on a per-plan basis. Currently the ODB supports colocating only the `post-deploy` or `pre-delete` errands.

For more information, see the Errands in the BOSH documentation.

To enable co-located errands for a plan, do the following steps.

1. Add each co-located errand job to the manifest as follows:
    - Add the errand in `service_deployment`.
    - Add the errand in the plan's `lifecycle_errands` configuration.
    - Set the instances the errand should run on in the `lifecycle_errands`.

    Below is an example manifest that includes a co-located post-deploy errand:

```
service_deployment:
  releases:
    - name: SERVICE-RELEASE
      version: SERVICE-RELEASE-VERSION
      jobs:
        - SERVICE-RELEASE-JOB
        - CO-LOCATED-POST-DEPLOY-ERRAND-JOB
service_catalog:
  plans:
    - name: CF-MARKETPLACE-PLAN-NAME
      lifecycle_errands:
        post_deploy:
          - name: CO-LOCATED-POST-DEPLOY-ERRAND-JOB
            instances:
```

```
              - SERVICE-RELEASE-JOB/0
          - name: NON-CO-LOCATED-POST-DEPLOY-ERRAND
      instance_groups:
        - name: NON-CO-LOCATED-POST-DEPLOY-ERRAND
          ...
        - name: SERVICE-RELEASE-JOB
          ...
```

Where `CO-LOCATED-POST-DEPLOY-ERRAND-JOB` is the co-located errand you want to run and `SERVICE-RELEASE-JOB/0` is the instances you want the errand to run on.

Create a pull request or raise an issue on the source for this page in GitHub

Broker and Service Management

This topic describes how to manage your broker and service plans.

Broker Management Errands

This section describes how to manage your broker with BOSH errands. You can run these errands using the BOSH CLI.

Register Broker

The `register-broker` errand registers the broker with Cloud Foundry and enables access to plans in the service catalog. Run this errand whenever the broker is re-deployed with new catalog metadata to update the Cloud Foundry catalog.

Add the Errand to the Manifest

To add the `register-broker` errand to the manifest, do the following:

1. Add the following instance groups to your manifest:

```
- name: register-broker
  lifecycle: errand
  instances: 1
  jobs:
    - name: register-broker
      release: ODB-RELEASE-NAME
      properties:
        broker_name: BROKER-NAME
        broker_uri: BROKER-URI # optional, only required when a route has been
registered
        disable_ssl_cert_verification: TRUE|FALSE # defaults to false
        enable_service_access: TRUE|FALSE # defaults to true
        default_access_org: ORG-NAME # optional, defaults to system
        cf:
          api_url: CF-API-URL
          admin_username: CF-API-ADMIN-USERNAME
          admin_password: CF-API-ADMIN-PASSWORD
  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [NETWORK]
  azs: [AZ]
```

```
# Optional. Add the cf_service_access property to the broker job:
- name: broker
  ...
  jobs:
    - name: broker
      ...
      properties:
        ...
        service_catalog:
          ...
          plans:
            - name: PLAN-NAME
              ...
              cf_service_access: ENABLE|DISABLE|MANUAL|ORG-RESTRICTED # optiona
l, defaults to enable
```

- If the `broker_uri` property is set, you must register a route for your broker with Cloud Foundry. See the Route Registration section for more details.
- When `enable_service_access: false` is set, the errand does not change service access for any plan.
- When `default_access_org` is set, the errand enables access to that org for any plans configured with `cf_service_access: org-restricted`.
- (Optional) Use the `cf_service_access` property to enable access to individual plans. Set the property on each plan in the broker job. The values accepted are the following:
  - `enable`: the errand enables access for that plan. This is the default value.
  - `disable`: the errand disables access for that plan.
  - `manual`: the errand does not modify service access for this plan.
  - `org-restricted`: the errand enables access for this plan to the `default_access_org` only.

  Only Cloud Foundry admin users can see plans with disabled service access. Org Managers and Space Managers cannot see these plans.

Run the Errand

To run the `register-broker` errand, do the following:

1. Run the command:

   ```
   bosh -d DEPLOYMENT-NAME run-errand register-broker
   ```

   Where:

   - `DEPLOYMENT-NAME` is the name of your deployment.

   For example:

   ```
   $ bosh -d cf run-errand register-broker
   ```

Delete All Service Instances

WARNING: Use extreme caution when running this errand. You should only use it when you want to totally destroy all of the on-demand service instances from Cloud Foundry.

The `delete-all-service-instances` errand deletes service instances of your broker's service offering in every org and space of Cloud Foundry. Because the errand uses the Cloud Controller API, it only deletes instances the Cloud Controller knows about.

The errand does not delete orphan BOSH deployments, which do not correspond to a known service instance. Orphan BOSH deployments should never happen, but in practice they do. Use the orphan-deployments errand to identify them.

The errand does the following:

1. Unbinds all apps from the service instances.
2. Deletes all service instances sequentially. Each service instance deletion includes:
    1. Running any pre-delete errands
    2. Deleting the BOSH deployment of the service instance
    3. Removing any ODB-managed secrets from CredHub
    4. Checking for instance deletion failure, which results in the errand failing immediately
3. Determines whether any instances have been created while the errand was running. If new instances are detected, the errand returns an error. In this case, Pivotal recommends running the errand again.

Add the Errand to the Manifest

To add the `delete-all-service-instances` errand to the manifest, do the following:

1. Add the following instance group to your manifest:

```
- name: delete-all-service-instances
  lifecycle: errand
  instances: 1
  jobs:
    - name: delete-all-service-instances
      release: ODB-RELEASE-NAME
      properties:
        polling_interval_seconds: INTERVAL-IN-SECONDS # defaults to 60
        polling_initial_offset_seconds: OFFSET-IN-SECONDS # defaults to 5

  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [{name: NETWORK}]
  azs: [AZ]
```

   Where:

   - `INTERVAL-IN-SECONDS`: The interval in seconds before a service instance is deleted.

- `OFFSET-IN-SECONDS`: The offset in seconds before polling Cloud Foundry to check if the instance has been deleted.

Notes:

- The `polling_interval_seconds` default is set to 60 seconds because the Cloud Controller itself polls the on-demand broker every 60 seconds. Setting your polling interval to anything lower than 60 seconds does not speed up the errand.
- The `polling_initial_offset_seconds` default is set to 5 seconds. In systems with more load, consider increasing the polling offset.

Run the Errand

To run the `delete-all-service-instances` errand, do the following:

1. Run the command:

```
bosh -d DEPLOYMENT-NAME run-errand \
delete-all-service-instances
```

Where:

- `DEPLOYMENT-NAME` is the name of your deployment.

For example:

```
$ bosh -d cf run-errand \
delete-all-service-instances
```

Deregister Broker

The `deregister-broker` errand deregisters a broker from Cloud Foundry. It requires that there are no existing service instances of your broker's service offering. If you run this errand with service instances remaining, it fails. Run the `delete-all-service-instances-and-deregister-broker` errand to remove remaining service instances and deregistering the broker. See Delete All Service Instances and Deregister Broker below.

Add the Errand to the Manifest

To add the `deregister-broker` errand to the manifest, do the following:

1. Add the following instance group to your manifest:

```
- name: deregister-broker
  lifecycle: errand
  instances: 1
  jobs:
    - name: deregister-broker
      release: ODB-RELEASE-NAME
      properties:
        broker_name: BROKER-NAME
```

```
  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [{name: SERVICE-NETWORK}]
  azs: [AZ]
```

Run the Errand

To run the `deregister-broker` errand, do the following:

1. Run the command:

   ```
   bosh -d DEPLOYMENT-NAME run-errand deregister-broker
   ```

   Where:

   - `DEPLOYMENT-NAME` is the name of your deployment.

   For example:

   ```
   $ bosh -d cf run-errand deregister-broker
   ```

Delete All Service Instances and Deregister Broker

WARNING: Use extreme caution when running this errand. You should only use it when you want to destroy all of the on-demand service instances and deregister the broker from Cloud Foundry.

The `delete-all-service-instances-and-deregister-broker` errand performs a similar operation to the errands delete-all-service-instances and deregister-broker.

This errand does the following:

1. Disables service access to the service offering for all orgs and spaces. The errand disables service access to ensure that new instances cannot be provisioned during the lifetime of the errand.
2. Unbinds all apps from the service instances.
3. Deletes all service instances sequentially. Each service instance deletion includes:
   1. Running any pre-delete errands
   2. Deleting the BOSH deployment of the service instance
   3. Removing any ODB-managed secrets from CredHub
   4. Checking for instance deletion failure, which results in the errand failing immediately
4. Determines whether any instances have been created while the errand was running. If new instances are detected, the errand returns an error. In this case, Pivotal recommends running the errand again.
5. Deregisters the broker from Cloud Foundry.

Add the Errand to the Manifest

To add the `delete-all-service-instances-and-deregister-broker` errand to the manifest, do the following:

1. Add the following instance group to your manifest:

```
- name: delete-all-service-instances-and-deregister-broker
  lifecycle: errand
  instances: 1
  jobs:
    - name: delete-all-service-instances-and-deregister-broker
      release: ODB-RELEASE-NAME
      properties:
        broker_name: BROKER-NAME
        polling_interval_seconds: INTERVAL-IN-SECONDS # defaults to 60
        polling_initial_offset_seconds: OFFSET-IN-SECONDS # defaults to 5

  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [{name: NETWORK}]
  azs: [AZ]
```

Where:

- `INTERVAL-IN-SECONDS`: The interval in seconds before a service instance is deleted.
- `OFFSET-IN-SECONDS`: The offset in seconds before polling Cloud Foundry to check if the instance has been deleted.

Notes:

- The `polling_interval_seconds` default is set to 60 seconds because the Cloud Controller itself polls the on-demand broker every 60 seconds. Setting your polling interval to anything lower than 60 seconds does not speed up the errand.
- The `polling_initial_offset_seconds` default is set to 5 seconds. In systems with more load, consider increasing the polling offset.

Run the Errand

To run the `delete-all-service-instances-and-deregister-broker` errand, do the following:

1. Run the command:

```
bosh -d DEPLOYMENT-NAME run-errand \
delete-all-service-instances-and-deregister-broker
```

Where:

- `DEPLOYMENT-NAME` is the name of your deployment.

For example:

```
$ bosh -d cf run-errand \
delete-all-service-instances-and-deregister-broker
```

Orphan Deployments

Note: The deployment for a service instance is orphaned when the BOSH deployment is still running but the service is no longer registered in Cloud Foundry.

The `orphan-deployments` errand lists service deployments that have no matching service instances in Cloud Foundry and returns the list to the operator.

Add the Errand to the Manifest

To add the `orphan-deployments` errand to the manifest, do the following:

1. Add the following instance group to your manifest:

```
- name: orphan-deployments
  lifecycle: errand
  instances: 1
  jobs:
  - name: orphan-deployments
    release: ODB-RELEASE-NAME
  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [{name: NETWORK}]
  azs: [AZ]
```

2. The `orphan-deployments` errand can be configured to use a Service Instances API. This might be required if your broker is deployed without Cloud Foundry. For more information, see Service Instances API.

Run the Errand

To run the `orphan-deployments` errand, do the following:

1. Run the command:

```
bosh -d DEPLOYMENT-NAME run-errand orphan-deployments
```

   Where:

   - `DEPLOYMENT-NAME` is the name of your deployment.

   For example:

```
$ bosh -d cf run-errand orphan-deployments
```

2. See if orphan deployments are present. If orphan deployments are present, the errand outputs a list that resembles the following:

```
Exit Code  10
Stdout     [{"deployment_name":"service-instance_bebbcf14-14ef-4eae-8fbd-656d15
f2b4b5"}]

Stderr     [orphan-deployments] 2019/04/03 14:56:02.489064 Orphan BOSH
```

```
deployments detected with no corresponding service instance in the
platform. Before deleting any deployment it is recommended to verify the
service instance no longer exists in the platform and any data is safe to
delete
```

Delete an Orphan Deployment

WARNING: Deleting the BOSH deployment destroys the service instance. Any data present is lost.

To delete an orphan deployment, do the following:

1. Run the command:

   ```
   bosh -d DEPLOYMENT-NAME delete-deployment
   ```

   Where:

   - `DEPLOYMENT-NAME` is the name of the orphaned deployment returned in the output of the errand.

   For example:

   ```
   bosh -d service-instance_aoeu39fgn-8125-05h2-9023-9vbxf7676f3 \
   delete-deployment
   ```

Recreate All Service Instances

Note: ODB only supports the `recreate-all-service-instances` errand in the following BOSH versions:

- 266.10.0 – 266.10.x
- 267.10.0 – 267.10.x
- 268.2.2 – 268.2.x
- 268.4.0 and later

Where x represents the latest version in that release line.

The `recreate-all-service-instances` errand recreates all service instance VMs managed by a broker. You might want use this errand when doing, for example, the following:

- Rotating the Ops Manager root certificate authority (CA). For more information about rotating CAs, see Rotate CAs and Leaf Certificates.
- A full restore of the platform during disaster recovery or migration.

Add the Errand to the Manifest

To add the `recreate-all-service-instances` errand to the manifest, do the following:

1. Add the following instance group to your manifest:

```
- name: recreate-all-service-instances
  lifecycle: errand
  instances: 1
  jobs:
    - name: recreate-all-service-instances
      release: ODB-RELEASE-NAME
      properties:
        polling_interval_seconds: POLLING-INTERVAL-IN-SECONDS # defaults to 60
        attempt_interval_seconds: ATTEMPT-INTERVAL-IN-SECONDS # defaults to 60
        attempt_limit: NUMBER-OF-ATTEMPTS # defaults to 5
  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [{name: NETWORK}]
  azs: [AZ]
```

You can configure the behavior of this errand using following properties:

Property	Description
`polling_interval_seconds`	This controls the wait between checking the status of successfully submitted BOSH recreate tasks.
`attempt_interval_seconds`	When there are BOSH tasks in progress on the service instance to recreate, the instance is put in a retry queue. This property controls the pause between retries.
`attempt_limit`	The number of times to check whether each instance is available to be recreated. After an instance reaches the limit, the errand terminates.

Run the Errand

To run the `recreate-all-service-instances` errand, do the following:

1. Run the command:

```
bosh -d DEPLOYMENT-NAME run-errand recreate-all-service-instances
```

Where:

- `DEPLOYMENT-NAME` is the name of your deployment.

For example:

```
$ bosh -d cf run-errand recreate-all-service-instances
```

Service Management

This section describes how to update, disable, and remove service plans. For how to upgrade the broker and service plans, see Upgrading.

Update the Broker

To modify the broker, do the following:

1. Make any necessary changes to the core broker configuration in the broker manifest. For more information about the core broker configuration, see Configure Your Broker.
2. Deploy the broker.

Update the Service Offering

To modify the service offering, do the following:

1. Change properties in the `service_catalog` of the broker manifest. For example, update the service metadata.
2. Change properties in the `service_deployment` of the broker manifest. For example, update the jobs used from a service release.
3. Deploy the broker.
4. Run the `register-broker` errand to update the Marketplace. For how to run the errand, see Register Broker above.
5. Run the `upgrade-all-service-instances` errand to apply updated plans to existing service instances. For how to run the errand, see Upgrade All Service Instances.

WARNING: When Cloud Foundry registers the broker, do not change `service_id` or `plan_id` for any plan.

Disable Service Plans

To disable access to a service plan, do the following:

1. Run the following command:

   ```
   cf disable-service-access SERVICE-NAME-FROM-CATALOG -p PLAN-NAME
   ```

   Where:

   - `SERVICE-NAME-FROM-CATALOG` is the name of the service from the service catalog.
   - `PLAN-NAME` is the name of the plan you want to disable.

   For example:

   ```
   cf disable-service-access my-service -p small
   ```

Note: When a plan has the property `cf_service_access: disable` in the `service_catalog` the Register Broker errand disables service access to that plan.

Remove Service Plans

You can remove service plans if there are no instances using the plan.

To remove a plan, do the following:

1. Remove the service plan from the broker manifest.

2. Run the `register-broker` errand to update the Marketplace. For more information about this errand, see Register Broker above.

WARNING: If any service instances remain on a plan that has been removed from the catalog, the On-Demand Service Broker fails to start.

Create a pull request or raise an issue on the source for this page in GitHub

Upgrading

This topic provides information about upgrading the on-demand broker, service offering, and service instance to Pivotal Cloud Foundry (PCF) Ops Manager operators and BOSH operators.

For product versions and upgrade paths, see Upgrade Planner.

Update Add-Ons to Run with Xenial Stemcell

If you are using a Xenial stemcell for your deployment and you are using any of the following BOSH add-ons, you must update the add-on definition to include the Xenial stemcell before you deploy your service:

- File Integrity Monitoring for PCF Add-on. For update instructions, see Updating FIM Add-on for PCF to Run with Xenial Stemcells.
- ClamAV for PCF Add-on. For update instructions, see Updating ClamAV Add-on for PCF to Run with Xenial Stemcells.
- IPsec for PCF Add-on. For update instructions, see Updating IPsec Add-on for PCF to Run with Xenial Stemcells.

Upgrade the Broker

To upgrade the broker, do the following:

1. Download a new version of the on-demand service broker BOSH release from Pivotal Network.
2. Upload the release to the BOSH Director by running:

```
bosh -e BOSH-DIRECTOR-NAME upload-release RELEASE-FILE-NAME.tgz
```

3. Make any necessary changes to the core broker configuration in the broker manifest. For more information about the core broker configuration, see Configure Your Broker.
4. Deploy the broker by running:

```
bosh -e BOSH-DIRECTOR-NAME -d DEPLOYMENT-NAME deploy DEPLOYMENT-MANIFEST.yml
```

Upgrade the Service Offering

The service offering is made of the following:

- Service catalog
- Service adapter BOSH release
- Service BOSH releases
- Service stemcells

To upgrade a service offering, do the following:

1. Make any changes to the service catalog in the broker manifest. For more information about the service catalog, see the Starter Snippet.
2. Upload any new service BOSH releases to the BOSH Director by running:

   ```
   bosh -e BOSH-DIRECTOR-NAME upload-release RELEASE-FILE-NAME.tgz
   ```

3. Make any changes to service releases in the broker manifest.
4. Upload any new service stemcells to the BOSH Director.

   ```
   bosh -e BOSH-DIRECTOR-NAME upload-stemcell STEMCELL-LOCATION
   ```

   Where `STEMCELL-LOCATION` is the path or URL of the stemcell.
5. Make any changes to the service stemcells in the `service_deployment` broker manifest.
6. Deploy the broker by running:

   ```
   bosh -e BOSH-DIRECTOR-NAME -d DEPLOYMENT-NAME deploy DEPLOYMENT-MANIFEST.yml
   ```

New service instances are created using the latest service offering. To upgrade all existing instances, you can run the upgrade-all-service-instances errand. See Upgrade All Service Instances below.

Warning: Until a service instance has been upgraded, `cf update-service` operations are blocked and an error is shown. For more information, see Cannot Update a Service Instance.

Upgrade All Service Instances

To upgrade all existing service instances after the service offering has been updated or upgraded, do the following:

1. Add the following instance group to your broker manifest:

   ```
   - name: upgrade-all-service-instances
     lifecycle: errand
     instances: 1
     jobs:
       - name: upgrade-all-service-instances
         release: ODB-RELEASE-NAME
         properties:
           canaries: NUMBER-OF-CANARIES # defaults to 0
           canary_selection_params:
             cf_org: ORG # specifying service instances to upgrade as canaries
   ```

```
          cf_space: SPACE # specifying service instances to upgrade as canaries
        max_in_flight: NUMBER-OF-PARALLEL-UPGRADES # defaults to 1
        polling_interval_seconds: POLLING-INTERVAL-IN-SECONDS # defaults to 60
        attempt_interval_seconds: ATTEMPT-INTERVAL-IN-SECONDS # defaults to 60
        attempt_limit: NUMBER-OF-ATTEMPTS # defaults to 5
  vm_type: VM-TYPE
  stemcell: STEMCELL
  networks: [{name: NETWORK}]
  azs: [AZ]
```

The errand properties allow fine-tuning of the behavior of the upgrade job:

Property	Description
`max_in_flight`	Sets the limit for the number of upgrades occurring concurrently. The number of simultaneous upgrades is limited by the number of available BOSH workers. See workers in the Cloud Foundry BOSH documentation. Set the `max_in_flight` value to lower than this limit to avoid over-saturating BOSH.
`canaries`	Sets the number of canary instances to upgrade first. If all canary instances upgrade, the remaining instances are upgraded. If a canary instance fails to upgrade or the `attempt_limit` is reached, the upgrade fails. No further instances are upgraded, and the errand exits with an error; however, all in-flight upgrades can complete. Canary instances are upgraded in parallel, respecting the `max_in_flight` value. **Note:** Canary instances are selected in a non-deterministic way using all available instances. If a selected instance is busy or was deleted, another instance is selected. If all instances are busy, the errand retries, respecting the `attempt_limit` and `attempt_interval_seconds`.
(Optional) `canary_selection_params`	Use this to specify an org and a space that you want canaries to be sourced from during an upgrade. If an org is specified, then a space must also be provided and vice versa. If `canaries` is specified, the broker upgrades that number of instances present in the org and space. If fewer instances are present than specified, the broker upgrades as many instances as possible in that org and space. **Note:** If `canary_selection_params` are specified and no instances exist in that org or space, no canaries are chosen. If other instances exist, the broker fails, alerting you to chose different selection criteria. If `canary_selection_params` is specified but empty, it is treated as if none was provided.
`polling_interval_seconds`	This controls the wait between checking the status of the successfully submitted BOSH upgrade job. If service instances have in-progress BOSH operations, upgrade requests are rejected and the errand queues those instances for a retry: - `attempt_interval_seconds` determines the time to wait between retrying upgrades. - `attempt_limit` sets the number of times these instances are retried for upgrade.

2. Deploy the broker manifest by running:

```
bosh -e BOSH-DIRECTOR-NAME -d DEPLOYMENT-NAME deploy DEPLOYMENT-MANIFEST.yml
```

3. Run the errand with the following command:

```
bosh -d DEPLOYMENT-NAME run-errand upgrade-all-service-instances
```

Note: The `upgrade-all-service-instances` errand triggers service instance lifecycle errands configured for the broker. For more information, see Service Instance Lifecycle Errands.

Service Instances API

The service instances API is used with an on-demand service broker without a connection to Cloud Foundry. It consists of an HTTP endpoint that the `orphan-deployments` and `upgrade-all-service-instances` errands can use to retrieve the list of instances known to the platform.

You can configure the API endpoint in the broker job.

WARNING: The Service Instances API is an advanced feature. Configuring a Service Instances API is not required, unless you are operating in very specific circumstances, such as running an on-demand service broker with no connection to Cloud Foundry.

You are required to provide an endpoint which satisfies the API requirements detailed below and also guarantees that all service instances which need to be upgraded are part of that response. If you configure an endpoint which does not satisfy these criteria, some service instances may become unusable.

Note: The Service Instances API definition is at v0.1.0 and should be considered subject to change.

General Request and Response

The API endpoint must handle the request below with the appropriate response.

Example Request:

```
curl -u BASIC-AUTH-USERNAME:BASIC-AUTH-PASSWORD \
-X GET \
http://SERVICE-INSTANCES-API-URL
```

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {"service_instance_id": GUID-1, "plan_id": "dedicated"},
  {"service_instance_id": GUID-2, "plan_id": "dedicated"},
  {"service_instance_id": GUID-3, "plan_id": "dedicated"},
  [...]
]
```

The Services Instances API returns other status codes such as `3xx`, `4xx`, or `5xx` for non-successful scenarios.

The plan IDs listed in the response must correspond to the plan IDs in each plan's definition in the ODB deployment manifest, not any other ID that may be assigned by the service controller.

Filtered Request and Response

The API endpoint must be able to provide a filtered response based on query parameters passed in the request.

The `canary_selection_params` property defines how to filter the canary instances from of a set of instances. If the on-demand service broker is configured to use `canary_selection_params`, The Service Instances API must respond with a filtered list of service instances when the `canary_selection_params` are passed as query parameters.

For example, if the `canary_selection_params` are configured as following:

```
canary_selection_params:
  cf_org: staging-org
  cf_space: staging-space
```

Then, the Service Instances API returns a filtered list of instances when `cf_org` and `cf_space` are passed as query parameters in the request.

Example Request:

```
curl -u BASIC-AUTH-USERNAME:BASIC-AUTH-PASSWORD \
-X GET \
http://SERVICE-INSTANCES-API-URL?cf_org=staging-org&cf_space=staging-space
```

Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {"service_instance_id": GUID-1, "plan_id": "dedicated"}
]
```

The instances in the filtered list are used as canary instances and upgraded before the rest. The number of canaries taken from this list can be configured by specifying the `canaries` property in the on-demand service broker manifest.

Configure the Broker to Use the Service Instances API

To configure the broker to use a Service Instance API provider for errands, update the broker's configuration in its deployment manifest to include the `service_instances_api` section:

```
- name: upgrade-all-service-instances
  ...
  jobs:
    - name: broker
      ...
```

```
    properties:
      ...
      # Add the following section:
      service_instances_api:
        # required:
        url: SERVICE-INSTANCES-API-URL
        # optional:
        root_ca_cert: ROOT-CA-CERT
        # optional - defaults to false:
        disable_ssl_cert_verification: TRUE|FALSE
        authentication:
          # required - currently the only supported authentication type:
          basic:
            username: USERNAME
            password: PASSWORD
...
```

Where `SERVICE-INSTANCES-API-URL` is the URL of the Service Instances API provider.

Create a pull request or raise an issue on the source for this page in GitHub

Security

This topic provides information about the API endpoints that the on-demand broker (ODB) accesses, and the minimum UAA authorities and corresponding permissions required for the ODB to operate correctly.

BOSH API Endpoints

ODB accesses the following BOSH API endpoints during the service instance lifecycle:

API endpoint	Examples of usage in ODB
`POST /deployments`	Create or update a service instance
`POST /deployments/DEPLOYMENT-NAME/errands/ERRAND-NAME/runs`	• Register or de-register the on-demand broker with the Cloud Controller • Run smoke tests
`GET /deployments/DEPLOYMENT-NAME`	Passed as argument to the service adapter for `generate-manifest` and `create-binding`
`GET /deployments/DEPLOYMENT-NAME/vms?format=full`	Passed as argument to the service adapter for `create-binding`
`DELETE /deployments/DEPLOYMENT-NAME`	Delete a service instance
`GET /tasks/TASK-ID/output?type=result`	• Check a task was successful (i.e. the exit code was zero) • Get list of VMs
`GET /tasks/TASK-ID`	Poll the BOSH Director until a task finishes, e.g. create, update, or delete a deployment
`GET /tasks?deployment=DEPLOYMENT-NAME`	Determine the last operation status and message for a service instance, e.g. 'create in progress'. Used when creating, updating, deleting service instances.

For information about BOSH API endpoints, see Director HTTP API in the BOSH documentation.

BOSH UAA Permissions

The actions that ODB needs to be able to perform are:

Modify:

- `bosh deploy`
- `bosh delete-deployment`
- `bosh run-errand`

Read only:

- `bosh deployments`
- `bosh vms`
- `bosh tasks`

The minimum UAA authority required by the BOSH Director to perform these actions is `bosh.teams.TEAM-NAME.admin`.

Note: A team admin cannot view or update the Director's cloud config, nor upload releases or stemcells.

For information on how to set up and use BOSH teams, see Using BOSH Teams in the BOSH documentation.

Unused BOSH permissions

The `bosh.teams.TEAM-NAME.admin` authority also allows the following actions, which currently are not used by ODB:

- `bosh start/stop/recreate`
- `bosh cck`
- `bosh logs`
- `bosh releases`
- `bosh stemcells`

PCF IPsec Add-On

Pivotal tested ODB with the IPsec Add-On for PCF and it appears to work. The tests excluded the BOSH Director itself from IPsec ranges because the BOSH add-on cannot be applied to BOSH itself.

For how to install the IPsec Add-On for PCF, see Installing the IPsec Add-on for PCF.

CF API Endpoints

ODB accesses the following CF API endpoints during the service instance lifecycle:

API endpoint	Examples of usage in ODB
`GET /v2/info`	Identify CF API version to determine feature compatibility & availability
`GET /v2/services`	List all services to find our own service based on defined unique ID rather than GUID
`GET /v2/services/SERVICE-GUID/service_plans`	Find registered service plans for ODB service e.g. for calculating plan quota usage
`GET /v2/service_brokers`	Find service broker metadata by name during broker deregistration
`DELETE /v2/service_brokers/SERVICE-BROKER-GUID`	Delete ODB service broker during broker deregister errand
`GET /v2/service_plans/SERVICE-PLAN-GUID`	Identify service plan when upgrading an instance to trigger any lifecycle errands
`PUT /v2/service_plans/SERVICE-PLAN-GUID`	Disable service access prior to service deletion
`GET /v2/service_plans/SERVICE-PLAN-GUID/service_instances`	Find service instances for given plan when determining global quota and running startup checks
`GET /v2/service_instances/SERVICE-INSTANCE-GUID`	Determine service instance state to check an operation is not in progress before triggering an upgrade
`DELETE /v2/service_instances/SERVICE-INSTANCE-GUID`	Deleting a service instance during delete all service instances errand
`GET /v2/service_instances/SERVICE-INSTANCE-GUID/service_bindings`	Finding bindings for given service instance during delete all service instances errand
`GET /v2/service_instances/SERVICE-INSTANCE-GUID/service_keys`	Finding service keys for given service instance during delete all service instances errand
`DELETE /v2/apps/APP-GUID/service_bindings/SERVICE-BINDING-GUID`	Unbinding a service instance during delete all service instances errand
`DELETE /v2/service_keys/SERVICE-KEY-GUID`	Deleting a service key during delete all service instances errand

For information about Cloud Foundry API endpoints, see the Cloud Foundry API documentation.

Cloud Foundry UAA Permissions

The actions that ODB needs to be able to perform are:

Modify:

- `cf enable-service-access`
- `cf disable-service-access`
- `cf create-service-broker`
- `cf delete-service-broker`
- `cf delete-service`
- `cf unbind-service`

- `cf delete-service-key`

Read only:

- `cf api`
- `cf marketplace`
- `cf service-brokers`
- `cf services`
- `cf service`
- `cf app`
- `cf service-keys`

The minimum UAA authority required by Cloud Foundry to perform these actions is `cloud_controller.admin`. Admin is required as many operations are required to perform against all of the on-demand service instances across a foundation, regardless of org and space.

Unused Cloud Foundry permissions

The Cloud Controller admin authority also allows the following actions, which currently are not used by ODB:

- `cf push`
- `cf delete`
- `cf start`
- `cf restart`
- `cf restage`
- `cf stop`
- `cf create-service-key`
- `cf create-user-provided-service`
- `cf update-user-provided-service`
- `cf run-task`
- `cf logs`
- `cf ssh`
- `cf scale`
- `cf events`
- Route and domain management
- Space management
- Org management
- CLI plugin management

Create a pull request or raise an issue on the source for this page in GitHub

Backup and Restore Considerations

This topic provides information about which components of an on-demand service have state and can be backed up.

On-Demand Service Broker

The on-demand service broker is stateless, so there is nothing to backup or restore.

On-Demand Service Instances

Service instances created by the on-demand service broker may have state that needs to be backed up, for example, data services.

It is the responsibility of the service author to provide documentation for the operator to backup and restore on-demand service instances. For a list of deliverables provided by the service author, see Service Author Deliverables.

Disaster Recovery

The on-demand service broker fetches the state of service instances and their deployments from the Cloud Foundry API and BOSH Director respectively. Therefore, to recover on-demand service instances in a disaster both the Cloud Controller database and BOSH Director database must be restored from a backup.

For how to backup and restore Pivotal Cloud Foundry (PCF) and BOSH, see Backing Up and Restoring Pivotal Cloud Foundry.

Create a pull request or raise an issue on the source for this page in GitHub

Data on Deployment Performance and Sizing

This topic contains test data on on-demand service instances managed in a Pivotal Cloud Foundry (PCF) environment.

Pivotal tested the on-demand broker (ODB) with 500 on-demand service instances using the example Kafka on-demand tile. We recorded how long it took to create, upgrade all, and delete all, with 50, 101, and 500 dedicated service instances. Setup and results are shown below.

Set up

Environment	
IaaS	Google Cloud Platform
PCF Operations Manager	v1.9.7
PCF Elastic Runtime	v1.9.13
Example Kafka On-Demand Tile	v0.15.1

BOSH Director Configuration	
Workers	3
Dedicated status worker	enabled
On-demand plan configuration	
Zookeeper VM type	small (1 CPU, 2GB RAM, 8GB Disk)
Kafka VM type	small (1 CPU, 2GB RAM, 8GB Disk)

Test

1. Upload the example Kafka on-demand tile.
2. Configure the on-demand plan.
3. Apply changes to install the on-demand service, ensuring that **Register On-Demand Broker** is checked.
4. Create N dedicated service instances using the cf CLI.
5. Make a change to the plan configuration.
6. Apply pending changes, ensuring that **Upgrade All On-Demand Service Instances** is checked.
7. Delete the tile and apply changes, ensuring that **Delete All On-Demand Service Instances** is checked.

Results

Durations presented in HH:MM:SS format.

Create	50	101	500
average create	00:01:02	00:01:03	00:01:02
total	00:51:28	01:45:40	08:33:37

Upgrade All	50	101	500
average upgarde	00:01:10	00:01:05	00:01:00
total	00:58:37	01:49:42	08:21:08

Delete All	50	101	500
average delete	00:05:09	00:05:04	0:05:00
total	04:17:38	08:31:10	41:38:26

These durations may vary for a number of reasons, for example:

- Number of BOSH director workers
- IaaS performance
- Network latency

- Service instance BOSH release(s)
- Service instance deployment configuration
- VM type of service instance
- Activity of Elastic Runtime
- Activity of BOSH Director

Notes

For create operations, the on-demand broker creates a BOSH deployment for each service instance. By default, the BOSH Director in Operations Manager v1.9 has three workers with a dedicated status worker, so only two workers are available to process deployment tasks. Therefore, only two service instances can be created at the same time.

For upgrade all and delete all operations, Operations Manager runs a BOSH errand. This errand task occupies a BOSH Director worker, leaving one worker available to upgrade, or delete deployments.

Create a pull request or raise an issue on the source for this page in GitHub

Troubleshooting On-Demand Services

Troubleshooting for BOSH Operators

This topic provides troubleshooting information for BOSH operators.

For more troubleshooting information, see Troubleshooting for Ops Manager Operators.

Administer Service Instances

Pivotal recommends that you use the BOSH CLI for administering the deployments created by the on-demand broker (ODB); for example for checking VMs, ssh, viewing logs. For more information on installing the BOSH CLI, see Install.

Pivotal discourages using the BOSH CLI to update or delete ODB service deployments as it causes `cf update-service` and `cf delete-service` operations to fail while the BOSH operation is in progress.

In addition, any changes you make to the deployment are reverted by `cf update-service` or by running the `upgrade-all-service-instances` errand. All updates to the service instances must be done using the `upgrade-all-service-instances` errand. For more information, see Upgrade All Service Instances.

Logs and Metrics

Logs

The ODB writes logs to a log file and to syslog.

The broker log contains error messages and non-zero exit codes returned by the service adapter, as well as the `stdout` and `stderr` streams of the adapter.

The log file is located at `/var/vcap/sys/log/broker/broker.log`. In syslog, logging is written with the tag `on-demand-service-broker`, under the facility `user`, with priority `info`.

If you want to forward syslog to a syslog aggregator, see Syslog Forwarding for Errand Logs below.

The ODB generates a UUID for each request and prefixes all the logs for that request, for example:

```
[on-demand-service-broker] [4d63080d-e038-45a3-85f9-93910f6b40b1] 2016/09/05 16:43:26.
123456 a valid UAA token was found in cache, will not obtain a new one
```

Note: The ODB's HTTP server and start up logs are not prefixed with a request ID.

All ODB logs have a UTC timestamp.

Syslog Forwarding for Errand Logs

If you want to forward your errand logs to a syslog aggregator, Pivotal recommends colocating syslog release with the errand job. For information, see the syslog release repository in GitHub.

Example manifest:

```
- name: delete-all-service-instances-and-deregister-broker
  lifecycle: errand
  ...
  jobs:
  - name: delete-all-service-instances-and-deregister-broker
    release: on-demand-service-broker
    ...
  - name: syslog_forwarder
    release: syslog
    properties:
      syslog:
        address: ((syslog.address))
        port: ((syslog.port))
        transport: udp
        forward_files: false
        custom_rule: |
          module(load="imfile" mode="polling")
          input(type="imfile"
                File="/var/vcap/sys/log/delete-all-service-instances-and-deregister-br
oker/errand.stdout.log"
                Tag="delete-all-service-instances-and-deregister-broker")
          input(type="imfile"
                File="/var/vcap/sys/log/delete-all-service-instances-and-deregister-br
oker/errand.stderr.log"
                Tag="delete-all-service-instances-and-deregister-broker")
```

Note: The errand is configured to redirect `stdout` and `stderr` to `/var/vcap/sys/log/ERRAND_NAME/errand.stdout.log` and `/var/vcap/sys/log/ERRAND_NAME/errand.stderr.log`. When configuring your errand, be careful to match the actual log file paths in the `custom_rule` section.

Metrics

If you have configured broker metrics, the broker emits metrics to the Loggregator Firehose. For how to do the configuration, see Configure Service Metrics.

You can consume these metrics by using the CF CLI Firehose plugin. See the firehose-plugin repository in GitHub.

Note: The broker must be registered with a Cloud Foundry in order for metrics to be successfully emitted. For how to register the broker, see Register Broker.

Service-level Metrics

The broker emits a metric indicating the total number of instances across all plans. In addition, if there is a global quota set for the service, a metric showing how much of that quota is remaining is emitted. Service-level metrics use the format shown below.

```
origin:"BROKER-DEPLOYMENT-NAME" eventType:ValueMetric timestamp:TIMESTAMP deployment:"
BROKER-DEPLOYMENT-NAME" job:"broker" index:"BOSH-JOB-INDEX" ip:"IP" valueMetric:<name:
"/on-demand-broker/SERVICE-OFFERING-NAME/total_instances" value:INSTANCE-COUNT unit:"c
ount" >
origin:"BROKER-DEPLOYMENT-NAME" eventType:ValueMetric timestamp:TIMESTAMP deployment:"
BROKER-DEPLOYMENT-NAME" job:"broker" index:"BOSH-JOB-INDEX>" ip:"IP" valueMetric:<name
:"/on-demand-broker/SERVICE-OFFERING-NAME/quota_remaining" value:QUOTA-REMAINING unit:
"count" >
```

Plan-level Metrics

For each service plan, the metrics report the total number of instances for that plan. If there is a quota set for the plan, the metrics also report how much of that quota is remaining. Plan-level metrics are emitted in the following format.

```
origin:"BROKER-DEPLOYMENT-NAME" eventType:ValueMetric timestamp:TIMESTAMP deployment:"
BROKER-DEPLOYMENT-NAME" job:"broker" index:"BOSH-JOB-INDEX" ip:"IP" valueMetric:<name:
"/on-demand-broker/SERVICE-OFFERING-NAME/PLAN-NAME/total_instances" value:INSTANCE-COU
NT unit:"count" >
origin:"BROKER-DEPLOYMENT-NAME" eventType:ValueMetric timestamp:TIMESTAMP deployment:"
BROKER-DEPLOYMENT-NAME" job:"broker" index:"BOSH-JOB-INDEX" ip:"IP" valueMetric:<name:
"/on-demand-broker/SERVICE-OFFERING-NAME/PLAN-NAME/quota_remaining" value:QUOTA-REMAIN
ING unit:"count" >
```

If `quota_remaining` is `0` then you need to increase your plan quota in the BOSH manifest.

Secure Binding Credentials

If you have configured secure binding credentials, the broker stores credentials on runtime CredHub. For more information, see Enable Secure Binding.

You can see and consume these credentials using the CredHub CLI. For more information, see the credHub-cli repository in GitHub.

Note: Usually, CredHub is not accessible from outside the Cloud Foundry network. Use the CredHub CLI from within the internal network, or connect using an

appropriate tunnel.

In failure scenarios, such as when CredHub is down or when the CredHub client credentials are wrong, the broker logs to the file at `/var/vcap/sys/log/broker/broker.log` where the root cause is generally given. For more information, see Logs above.

Common Causes of Errors

The following are some reasons that you might get an error:

- CredHub is down / wrong CredHub URL / cannot access URL
- Wrong credentials to access CredHub
- Problem with CA certs for CredHub or UAA
- Binding credentials in an exotic format (the broker only accepts string and string map credentials)

Identify Deployments in BOSH

There is a one-to-one mapping between the service instance ID from Cloud Foundry and the deployment name in BOSH. The convention is that the BOSH deployment name is the service instance ID prefixed by `service-instance_`. To identify the BOSH deployment for a service instance you can do the following:

1. Determine the GUID of the service. Run the following command:

   ```
   cf service --guid SERVICE-NAME
   ```

   Where `SERVICE-NAME` is the name of your service.

   For example:

   ```
   $ cf service --guid my-service
   ```

   Record the GUID in the output of the command.

2. Identify your deployment. Run `bosh deployments` and look for `service-instance_GUID`.
3. (Optional) Get current tasks for your deployment. Run the following command:

   ```
   bosh tasks -d service-instance_GUID
   ```

   Where `GUID` is the GUID for your service instance, which you retrieved above.

   For example:

   ```
   $ bosh tasks -d \
   service-instance_30d4a67f-d220-4d06-9989-58a976b86b35
   ```

Identify Tasks in BOSH

Most operations on the on demand service broker API are implemented by launching BOSH tasks. If an operation fails, it may be useful to investigate the corresponding BOSH task. For more information about BOSH tasks, see Tasks in the BOSH documentation.

To identify tasks in BOSH, do the following:

1. Determine the ID of the service for which an operation failed. Run the following command:

   ```
   cf service --guid SERVICE-NAME
   ```

   Where `SERVICE-NAME` is the name of your service.

   For example:

   ```
   $ cf service --guid my-service
   ```

   Record the GUID in the output of the command.

2. SSH on to the service broker VM. Run the following command:

   ```
   bosh -d BROKER-DEPLOYMENT-NAME ssh
   ```

   Where `BROKER-DEPLOYMENT-NAME` is the name of your broker deployment.

   For example:

   ```
   $ bosh -d my-broker ssh
   ```

3. In the broker log, look for lines relating to the service, identified by the service ID. Lines recording the starting and finishing of BOSH tasks also have the BOSH task ID:

   ```
   on-demand-service-broker: [on-demand-service-broker] [4d63080d-e038-45a3-85f9-9
   3910f6b40b1] 2016/04/13 09:01:50.793965 Bosh task id for Create instance 30d4a6
   7f-d220-4d06-9989-58a976b86b35 was 11470
   on-demand-service-broker: [on-demand-service-broker] [4d63080d-e038-45a3-85f9-9
   3910f6b40b1] 2016/04/13 09:06:55.793976 task 11470 success creating deployment
   for instance 30d4a67f-d220-4d06-9989-58a976b86b35: create deployment

   on-demand-service-broker: [on-demand-service-broker] [8bf5c9f6-7acd-4ab4-9214-3
   63a6f6bef79] 2016/04/13 09:16:20.795035 Bosh task id for Update instance 30d4a6
   7f-d220-4d06-9989-58a976b86b35 was 11473
   on-demand-service-broker: [on-demand-service-broker] [8bf5c9f6-7acd-4ab4-9214-3
   63a6f6bef79] 2016/04/13 09:17:20.795181 task 11473 success updating deployment
   for instance 30d4a67f-d220-4d06-9989-58a976b86b35: create deployment

   on-demand-service-broker: [on-demand-service-broker] [af6fab15-c95e-438b-aa6b-b
   c4329d4154f] 2016/04/13 09:17:52.803824 Bosh task id for Delete instance 30d4a6
   7f-d220-4d06-9989-58a976b86b35 was 11474
   on-demand-service-broker: [on-demand-service-broker] [af6fab15-c95e-438b-aa6b-b
   c4329d4154f] 2016/04/13 09:19:56.803938 task 11474 success deleting deployment
   for instance 30d4a67f-d220-4d06-9989-58a976b86b35: delete deployment service-in
   stance_30d4a67f-d220-4d06-9989-58a976b86b35
   ```

4. Use the task ID to obtain the task log from BOSH, adding flags such as `--debug` or `--cpi` as necessary. For example:

```
$ bosh task 11470
```

Identify Issues When Connecting to BOSH or UAA

The ODB interacts with the BOSH Director to provision and deprovision instances, and is authenticated through the Director's UAA. For an example configuration, see kafka-example-service-adapter-release in GitHub.

If BOSH or UAA are configured incorrectly in the broker's manifest, then error messages are displayed in the broker's log. These messages indicate whether the issue is caused by an unreachable destination or bad credentials.

For example:

```
on-demand-service-broker: [on-demand-service-broker]
[575afbc1-b541-481d-9cde-b3d3e67e87bf] 2016/05/18 15:56:40.100579
Error authenticating (401): {"error":"unauthorized","error_description":
"Bad credentials"}, ensure that properties.BROKER-JOB.bosh.authentication.uaa is
correct and try again.
```

List Service Instances

ODB persists the list of ODB-deployed service instances and provides an endpoint to retrieve them. This endpoint requires basic authentication.

During disaster recovery, you can use this endpoint to assess the situation.

Request:

```
GET http://USERNAME:PASSWORD@ON-DEMAND-BROKER-IP:8080/mgmt/service_instances
```

Response:

```
200 OK
```

Example JSON body:

```
[
  {
    "instance_id": "4d19462c-33cf-11e6-91cc-685b3585cc4e",
    "plan_id": "60476620-33cf-11e6-a841-685b3585cc4e",
    "bosh_deployment_name": "service-instance_4d19462c-33cf-11e6-91cc-685b3585cc4e"
  },
  {
    "instance_id": "57014734-33cf-11e6-ba8d-685b3585cc4e",
    "plan_id": "60476620-33cf-11e6-a841-685b3585cc4e",
    "bosh_deployment_name": "service-instance_57014734-33cf-11e6-ba8d-685b3585cc4e"
  }
]
```

List Orphan Deployments

ODB provides an endpoint that compares the list of service instance deployments against the service instances registered in Cloud Foundry. When called, the endpoint returns a list of orphaned

deployments, if any are present.

This endpoint is exercised in the `orphan-deployments` errand. For information about this errand, see Orphan Deployments. To call this endpoint without running the errand, use `curl`.

Request:

```
GET http://USERNAME:PASSWORD@ON-DEMAND-BROKER-IP:8080/mgmt/orphan_deployments
```

Response:

```
200 OK
```

Example JSON body:

```
[
  {
      "deployment_name": "service-instance_d482abd3-8051-48d2-8067-9ccdf02327f3"
  }
]
```

Knowledge Base (Community)

Find the answer to your question and browse product discussions and solutions by searching the VMware Tanzu Knowledge Base.

File a Support Ticket

You can file a ticket with Support. Be sure to provide the error message from `cf service YOUR-SERVICE-INSTANCE`.

To expedite troubleshooting, provide your service broker logs and your service instance logs. If your `cf service YOUR-SERVICE-INSTANCE` output includes a `task-id`, provide the BOSH task output.

Create a pull request or raise an issue on the source for this page in GitHub

Troubleshooting for Ops Manager Operators

This topic provides information for operators about troubleshooting on-demand services.

How to Retrieve a Service Instance GUID

You need the GUID of your service instance to run some BOSH commands. To retrieve the GUID, run the command:

```
cf service SERVICE-INSTANCE-NAME --guid
```

If you do not know the name of the service instance, run `cf services` to see a listing of all service instances in the space. The service instances are listed in the name column.

Troubleshoot Errors

This section provides information about how to troubleshoot specific errors or error messages.

Failed Installation

Symptom	On-Demand Services SDK fails to install.

Cause	Reasons for a failed installation include:
	• Certificate issues: The on-demand broker (ODB) requires valid certificates.
	• Deploy fails. This could be due to a variety of reasons.
	• Networking problems:
	  ◦ Cloud Foundry cannot reach the On-Demand Services SDK broker
	  ◦ Cloud Foundry cannot reach the service instances
	  ◦ The service network cannot access the BOSH director
	• The Register broker errand fails.
	• The smoke test errand fails.
	• Resource sizing issues: These occur when the resource sizes selected for a given plan are less than On-Demand Services SDK requires to function.
	• Other service-specific issues.

Solution	To troubleshoot: • Certificate issues: Ensure that your certificates are valid and generate new ones if necessary. To generate new certificates, contact Support. • Deploy fails: View the logs using Ops Manager to determine why the deploy is failing. • Networking problems: For how to troubleshoot, see Networking problems. • Register broker errand fails: For how to troubleshoot, see Register broker errand. • Resource sizing issues: Check your resource configuration in Ops Manager and ensure that the configuration matches that recommended by the service.

Cannot Create or Delete Service Instances

Symptom

If developers report errors such as:

```
Instance provisioning failed: There was a problem completing your r
equest. Please contact your operations team providing the following
 information: service: redis-acceptance, service-instance-guid: ae9
e232c-0bd5-4684-af27-1b08b0c70089, broker-request-id: 63da3a35-24aa
-4183-aec6-db8294506bac, task-id: 442, operation: create
```

Cause

Reasons include:

- Problems with the deployment manifest
- Authentication errors
- Network errors
- Quota errors

Solution

To troubleshoot:

1. If the BOSH error shows a problem with the deployment manifest, open the manifest in a text editor to inspect it.
2. To continue troubleshooting, Log in to BOSH and target the On-Demand Services SDK instance using the instructions on parsing a Cloud Foundry error message.
3. Retrieve the BOSH task ID from the error message and run the following command:

   ```
   bosh task TASK-ID
   ```

4. If you need more information, access the broker logs and use the `broker-request-id` from the error message above to search the logs for more information. Check for:
   - Authentication errors
   - Network errors
   - Quota errors

Broker Request Timeouts

Symptom

If developers report errors such as:

```
Server error, status code: 504, error code: 10001, message: The req
uest to the service broker timed out: https://BROKER-URL/v2/service
_instances/e34046d3-2379-40d0-a318-d54fc7a5b13f/service_bindings/aa
635a3b-ef6d-41c3-a23f-55752f3f651b
```

Cause

Cloud Foundry might not be connected to the service broker, or there might be a large number of queued tasks.

Solution	To troubleshoot: 1. Confirm that Cloud Foundry (CF) is connected to the service broker. 2. Check the BOSH queue size: 1. Log in to BOSH as an admin. 2. Run `bosh tasks` If there are a large number of queued tasks, the system may be under too much load. BOSH is configured with two workers and one status worker, which might not be sufficient resources for the level of load. 3. If the task queue is long, advise app developers to try again once the system is under less load.

Instance Does Not Exist

Symptom	If developers report errors such as: `Server error, status code: 502, error code: 10001, message: Service broker error: instance does not exist``
Cause	The instance might have been deleted.

Solution	To troubleshoot:
	1. Confirm that the On-Demand Services SDK instance exists in BOSH and obtain the GUID CF by running:
	`cf service MY-INSTANCE --guid`
	2. Using the GUID obtained above, run:
	`bosh -d service-instance_GUID vms`
	If the BOSH deployment is not found, it has been deleted from BOSH. Contact Support for further assistance.

Cannot Bind to or Unbind from Service Instances

Symptom	If developers report errors such as: `Server error, status code: 502, error code: 10001, message: Service broker error: There was a problem completing your request. Please contact your operations team providing the following information: service: example-service, service-instance-guid: 8d69de6c-88c6-4283-b8bc-1c46103714e2, broker-request-id: 15f4f87e-200a-4b1a-b76c-1c4b6597c2e1, operation: bind`
Cause	This might be due to authentication or network errors.

Solution	To find out the exact issue with the binding process:
	1. Access the service broker logs.
	2. Search the logs for the `broker-request-id` string listed in the error message above.
	3. Check for:
	  - Authentication errors
	  - Network errors
	4. Contact Support for further assistance if you are unable to resolve the problem.

Cannot Connect to a Service Instance

Symptom	Developers report that their app cannot use service instances that they have successfully created and bound.
Cause	The error might originate from the service or be network related.
Solution	To solve this issue, ask the user to send application logs that show the connection error. If the error originates from the service, then follow On-Demand Services SDK-specific instructions. If the issue appears to be network-related, then: 1. Check that application security groups are configured correctly. Access should be configured for the service network that the tile is deployed to. 2. Ensure that the network the PAS tile is deployed to has network access to the service network. You can find the network definition for this service network in the BOSH Director tile. 3. In Ops Manager go into the service tile and see the service network that is configured in the networks tab. 4. In Ops Manager go into the PAS tile and see the network it is assigned to. Make sure that these networks can access each other.

Cannot Update a Service Instance

Symptom	If developers report errors such as the following when trying to run `cf-update-service`:
	`FAILED`
	`Server error, status code: 502, error code: 10001, message: Service broker error: Service cannot be updated at this time, please try again later or contact your operator for more information.`

Cause	Their service instance might not be running the latest service offering.
Solution	Operators must run the `upgrade-all-service-instances` errand after upgrading to ensure all existing service instances are upgraded to the latest service offering. See Upgrade All Service Instances. App developers cannot upgrade individual service instances to the latest service offering. They cannot set parameters or change plan until you upgrade their service instances.

Upgrade All Service Instances Errand Fails

Symptom	The `upgrade-all-service-instances` errand fails.
Cause	There might be a problem with a particular instance.
Solution	To troubleshoot: 1. Look at the errand output in the Ops Manager log. 2. If an instance has failed to upgrade, debug and fix it before running the errand again to prevent any failure issues from spreading to other on-demand instances. 3. After the Ops Manager log no longer lists the deployment as `failing`, re-run the errand to upgrade the rest of the instances.

Missing Logs and Metrics

Symptom	No logs are being emitted by the on-demand broker.
Cause	Syslog might not be configured correctly, or you might have network access issues.

Solution	To troubleshoot:
	1. Ensure you have configured syslog for the tile.
	2. Check that your syslog forwarding address is correct in Ops Manager.
	3. Ensure that you have network connectivity between the networks that the tile is using and the syslog destination. If the destination is external, you need to use the public ip VM extension feature available in your Ops Manager tile configuration settings.
	4. Verify that Loggregator is emitting metrics:
	1. Install the `cf log-stream` plugin. For instructions, see the Log Stream CLI Plugin GitHub repository.
	2. Find the GUID for your service instance by running:
	`cf service SERVICE-INSTANCE --guid`
	3. Find logs from your service instance by running:
	`cf log-stream | grep "SERVICE-GUID"`
	4. If no metrics appear within five minutes, verify that the broker network has access to the Loggregator system on all required ports.
	5. If you are unable to resolve the issue, contact Support.

Unable to Render Templates for Job CredHub

Symptom	You receive an error similar to the following:
	
	`Task 54 | 19:40:25 | Creating missing vms: credhub/03CCE517-F40D-42C8-B758-606FC042A8D4 (0) (00:00:53)`
	`L Error: - Unable to render templates for job 'credhub'. Errors are:`
	`- Failed to fetch variable '/opsmgr/cf-f0052ab642e7684aa41d/credhub_key_encryption_passwords/0/key' with id '703CCE517-F40D-42C8-B758-606FC042A8D4' from config server: Invalid JSON response`

Cause	This might be due to an issue with BOSH CredHub v1.9.11 and v2.1.5.
Solution	Use CredHub v1.9.12 or later (included in Ops Manager v2.2.22 or later and v2.3.16 or later) or CredHub v2.1.6 (included in Ops Manager v2.4.10 or later and v2.5.3 or later).

Troubleshoot Components

This section provides information about troubleshooting on-demand broker components.

BOSH Problems

Large BOSH Queue

On-demand service brokers add tasks to the BOSH request queue, which can back up and cause delay under heavy loads. An app developer who requests a new On-Demand Services SDK instance sees `create in progress` in the Cloud Foundry Command Line Interface (cf CLI) until BOSH processes the queued request.

Ops Manager currently deploys two BOSH workers to process its queue. Future versions of Ops Manager will let users configure the number of BOSH workers.

Configuration

Service Instances in Failing State

The VM or Disk type that you configured in the plan page of the tile in Ops Manager might not be large enough for the On-Demand Services SDK service instance to start. See tile-specific guidance on resource requirements.

Authentication

UAA Changes

If you have rotated any UAA user credentials then you may see authentication issues in the service broker logs.

To resolve this, redeploy the On-Demand Services SDK tile in Ops Manager. This provides the broker with the latest configuration.

Note: You must ensure that any changes to UAA credentials are reflected in the Ops Manager `credentials` tab of the Pivotal Application Service tile.

Networking

Common issues with networking include:

Issue	Solution
Latency when connecting to the On-Demand Services SDK service instance to create or delete a binding.	Try again or improve network performance.
Firewall rules are blocking connections from the On-Demand Services SDK service broker to the service instance.	Open the On-Demand Services SDK tile in Ops Manager and check the two networks configured in the **Networks** pane. Ensure that these networks allow access to each other.
Firewall rules are blocking connections from the service network to the BOSH director network.	Ensure that service instances can access the Director so that the BOSH agents can report in.
Apps cannot access the service network.	Configure Cloud Foundry application security groups to allow runtime access to the service network.
Problems accessing BOSH's UAA or the BOSH director.	Follow network troubleshooting and check that the BOSH director is online

Validate Service Broker Connectivity to Service Instances

To validate connectivity, do the following:

1. View the BOSH deployment name for your service broker by running:

    ```
    bosh deployments
    ```

2. SSH into the On-Demand Services SDK service broker by running:

    ```
    bosh -d DEPLOYMENT-NAME ssh
    ```

3. If no BOSH `task-id` appears in the error message, look in the broker log using the `broker-request-id` from the task.

Validate App Access to Service Instance

Use `cf ssh` to access to the app container, then try connecting to the On-Demand Services SDK service instance using the binding included in the `VCAP_SERVICES` environment variable.

Quotas

Plan Quota Issues

If developers report errors such as:

```
Message: Service broker error: The quota for this service plan has been exceeded.
Please contact your Operator for help.
```

1. Check your current plan quota.

2. Increase the plan quota.
3. Log in to Ops Manager.
4. Reconfigure the quota on the plan page.
5. Deploy the tile.
6. Find who is using the plan quota and take the appropriate action.

Global Quota Issues

If developers report errors such as:

```
Message: Service broker error: The quota for this service has been exceeded.
Please contact your Operator for help.
```

1. Check your current global quota.
2. Increase the global quota.
3. Log in to Ops Manager.
4. Reconfigure the quota on the on-demand settings page.
5. Deploy the tile.
6. Find out who is using the quota and take the appropriate action.

Failing Jobs and Unhealthy Instances

To determine whether there is an issue with the On-Demand Services SDK deployment:

1. Inspect the VMs by running:

   ```
   bosh -d service-instance_GUID vms --vitals
   ```

2. For additional information, run:

   ```
   bosh -d service-instance_GUID instances --ps --vitals
   ```

If the VM is failing, follow the service-specific information. Any unadvised corrective actions (such as running BOSH `restart` on a VM) can cause issues in the service instance.

Techniques for Troubleshooting

This section provides general techniques for troubleshooting, which might include the following:

- Interacting with the on-demand service broker
- Interacting with on-demand service instance BOSH deployments
- Performing general maintenance and housekeeping tasks

Parse a Cloud Foundry (CF) Error Message

Failed operations (create, update, bind, unbind, delete) result in an error message. You can retrieve the error message later by running the cf CLI command `cf service INSTANCE-NAME`.

```
$ cf service myservice

Service instance: myservice
Service: super-db
Bound apps:
Tags:
Plan: dedicated-vm
Description: Dedicated Instance
Documentation url:
Dashboard:

Last Operation
Status: create failed
Message: Instance provisioning failed: There was a problem completing your request.
     Please contact your operations team providing the following information:
     service: redis-acceptance,
     service-instance-guid: ae9e232c-0bd5-4684-af27-1b08b0c70089,
     broker-request-id: 63da3a35-24aa-4183-aec6-db8294506bac,
     task-id: 442,
     operation: create
Started: 2017-03-13T10:16:55Z
Updated: 2017-03-13T10:17:58Z
```

Use the information in the `Message` field to debug further. Provide this information to Support when filing a ticket.

The `task-id` field maps to the BOSH task ID. For more information on a failed BOSH task, use the `bosh task TASK-ID`.

The `broker-request-guid` maps to the portion of the On-Demand Broker log containing the failed step. Access the broker log through your syslog aggregator, or access BOSH logs for the broker by typing `bosh logs broker 0`. If you have more than one broker instance, repeat this process for each instance.

Access Broker and Instance Logs and VMs

Before following the procedures below, log in to the cf CLI and the BOSH CLI.

Access Broker Logs and VM(s)

You can access logs using Ops Manager by clicking on the **Logs** tab in the tile and downloading the broker logs.

To access logs using the BOSH CLI, do the following:

1. Identify the on-demand broker (ODB) deployment by running the following command:

   ```
   bosh deployments
   ```

2. View VMs in the deployment by running the following command:

   ```
   bosh -d DEPLOYMENT-NAME instances
   ```

3. SSH onto the VM by running the following command:

```
bosh -d DEPLOYMENT-NAME ssh
```

4. Download the broker logs by running the following command:

```
bosh -d DEPLOYMENT-NAME logs
```

The archive generated by BOSH includes the following logs:

Log Name	Description
broker.stdout.log	Requests to the on-demand broker and the actions the broker performs while orchestrating the request (e.g. generating a manifest and calling BOSH). Start here when troubleshooting.
bpm.log	Control script logs for starting and stopping the on-demand broker.
post-start.stderr.log	Errors that occur during post-start verification.
post-start.stdout.log	Post-start verification.
drain.stderr.log	Errors that occur while running the drain script.

Access Service Instance Logs and VMs

1. To target an individual service instance deployment, retrieve the GUID of your service instance with the following cf CLI command:

```
cf service MY-SERVICE --guid
```

2. To view VMs in the deployment, run the following command:

```
bosh -d service-instance_GUID instances
```

3. To SSH into a VM, run the following command:

```
bosh -d service-instance_GUID ssh
```

4. To download the instance logs, run the following command:

```
bosh -d service-instance_GUID logs
```

Run Service Broker Errands to Manage Brokers and Instances

From the BOSH CLI, you can run service broker errands that manage the service brokers and perform mass operations on the service instances that the brokers created. These service broker errands include:

- `register-broker` registers a broker with the Cloud Controller and lists it in the Marketplace.
- `deregister-broker` deregisters a broker with the Cloud Controller and removes it from the Marketplace.

- `upgrade-all-service-instances` upgrades existing instances of a service to its latest installed version.
- `delete-all-service-instances` deletes all instances of service.
- `orphan-deployments` detects “orphan” instances that are running on BOSH but not registered with the Cloud Controller.

To run an errand, run the following command:

```
bosh -d DEPLOYMENT-NAME run-errand ERRAND-NAME
```

For example:

```
bosh -d my-deployment run-errand deregister-broker
```

Register Broker

The `register-broker` errand does the following:

- Registers the service broker with Cloud Controller.
- Enables service access for any plans that are enabled on the tile.
- Disables service access for any plans that are disabled on the tile.
- Does nothing for any plans that are set to manual on the tile.

You should run this errand whenever the broker is re-deployed with new catalog metadata to update the Marketplace.

Plans with disabled service access are only visible to admin Cloud Foundry users. Non-admin Cloud Foundry users, including Org Managers and Space Managers, cannot see these plans.

Deregister Broker

This errand deregisters a broker from Cloud Foundry.

The errand does the following:

- Deletes the service broker from Cloud Controller
- Fails if there are any service instances, with or without bindings

Use the Delete All Service Instances errand to delete any existing service instances.

To run the errand, run the following command:

```
bosh -d DEPLOYMENT-NAME run-errand deregister-broker
```

Upgrade All Service Instances

The `upgrade-all-service-instances` errand does the following:

- Collects all of the service instances that the on-demand broker has registered.
- Issues an upgrade command and deploys the a new manifest to the on-demand broker for

each service instance.

- Adds to a retry list any instances that have ongoing BOSH tasks at the time of upgrade.
- Retries any instances in the retry list until all instances are upgraded.

When you make changes to the plan configuration, the errand upgrades all the On-Demand Services SDK service instances to the latest version of the plan.

If any instance fails to upgrade, the errand fails immediately. This prevents systemic problems from spreading to the rest of your service instances.

Delete All Service Instances

This errand uses the Cloud Controller API to delete all instances of your broker's service offering in every Cloud Foundry org and space. It only deletes instances the Cloud Controller knows about. It does not delete orphan BOSH deployments.

Note: Orphan BOSH deployments do not correspond to a known service instance. While rare, orphan deployments can occur. Use the `orphan-deployments` errand to identify them.

The `delete-all-service-instances` errand does the following:

1. Unbinds all apps from the service instances.
2. Deletes all service instances sequentially. Each service instance deletion includes:
    1. Running any pre-delete errands
    2. Deleting the BOSH deployment of the service instance
    3. Removing any ODB-managed secrets from BOSH CredHub
    4. Checking for instance deletion failure, which results in the errand failing immediately
3. Determines whether any instances have been created while the errand was running. If new instances are detected, the errand returns an error. In this case, VMware recommends running the errand again.

Warning: Use extreme caution when running this errand. You should only use it when you want to totally destroy all of the on-demand service instances in an environment.

To run the errand, run the following command:

```
bosh -d service-instance_GUID delete-deployment
```

Detect Orphaned Instances Service Instances

A service instance is defined as "orphaned" when the BOSH deployment for the instance is still running, but the service is no longer registered in Cloud Foundry.

The `orphan-deployments` errand collates a list of service deployments that have no matching service

instances in Cloud Foundry and return the list to the operator. It is then up to the operator to remove the orphaned BOSH deployments.

To run the errand, run the following command:

```
bosh -d DEPLOYMENT-NAME run-errand orphan-deployments
```

If orphan deployments exist—The errand script does the following:

- Exit with exit code 10
- Output a list of deployment names under a `[stdout]` header
- Provide a detailed error message under a `[stderr]` header

For example:

```
[stdout]
[{"deployment\_name":"service-instance\_80e3c5a7-80be-49f0-8512-44840f3c4d1b"}]

[stderr]
Orphan BOSH deployments detected with no corresponding service instance in Cloud Found
ry. Before deleting any deployment it is recommended to verify the service instance no
 longer exists in Cloud Foundry and any data is safe to delete.

Errand 'orphan-deployments' completed with error (exit code 10)
```

These details will also be available through the BOSH `/tasks/` API endpoint for use in scripting:

```
$ curl 'https://bosh-user:bosh-password@bosh-url:25555/tasks/task-id/output?type=resul
t' | jq .
{
  "exit_code": 10,
  "stdout": "[{"deployment_name":"service-instance_80e3c5a7-80be-49f0-8512-44840f3c4d1
b"}]\n",
  "stderr": "Orphan BOSH deployments detected with no corresponding service instance i
n Cloud Foundry. Before deleting any deployment it is recommended to verify the servic
e instance no longer exists in Cloud Foundry and any data is safe to delete.\n",
  "logs": {
    "blobstore_id": "d830c4bf-8086-4bc2-8c1d-54d3a3c6d88d"
  }
}
```

If no orphan deployments exist—The errand script does the following:

- Exit with exit code 0
- Stdout will be an empty list of deployments
- Stderr will be `None`

```
[stdout]
[]

[stderr]
None

Errand 'orphan-deployments' completed successfully (exit code 0)
```

If the errand encounters an error during running—The errand script does the following:

- Exit with exit 1
- Stdout will be empty
- Any error messages will be under stderr

To clean up orphaned instances, run the following command on each instance:

WARNING: Running this command may leave IaaS resources in an unusable state.

```
bosh delete-deployment service-instance_SERVICE-INSTANCE-GUID
```

Get Admin Credentials for a Service Instance

To retrieve the admin credentials for a service instance from BOSH CredHub:

1. Use the cf CLI to determine the GUID associated with the service instance for which you want to retrieve credentials by running:

   ```
   cf service SERVICE-INSTANCE-NAME --guid
   ```

   For example:

   ```
   $ cf service my-service-instance --guid

   12345678-90ab-cdef-1234-567890abcdef
   ```

   If you do not know the name of the service instance, you can list service instances in the space with `cf services`.

2. Follow the steps in Gather Credential and IP Address Information and Log In to the Ops Manager VM with SSH of *Advanced Troubleshooting with the BOSH CLI* to SSH into the Ops Manager VM.

3. From the Ops Manager VM, log in to your BOSH Director with the BOSH CLI. See Authenticate with the BOSH Director VM in *Advanced Troubleshooting with the BOSH CLI*.

4. Find the values for `BOSH_CLIENT` and `BOSH_CLIENT_SECRET`:

   1. In the Ops Manager Installation Dashboard, click the **BOSH Director** tile.
   2. Click the **Credentials** tab.
   3. In the **BOSH Director** section, click the link to the **BOSH Commandline Credentials** .
   4. Record the values for `BOSH_CLIENT` and `BOSH_CLIENT_SECRET`.

5. Set the API target of the CredHub CLI to your BOSH CredHub server by running:

   ```
   credhub api https://BOSH-DIRECTOR-IP:8844 \
         --ca-cert=/var/tempest/workspaces/default/root_ca_certificate
   ```

   Where `BOSH-DIRECTOR-IP` is the IP address of the BOSH Director VM.

For example:

```
$ credhub api https://10.0.0.5:8844 \
      --ca-cert=/var/tempest/workspaces/default/root_ca_certificate
```

6. Log in to CredHub by running:

```
credhub login \
    --client-name=BOSH-CLIENT \
    --client-secret=BOSH-CLIENT-SECRET
```

For example:

```
$ credhub login \
      --client-name=credhub \
      --client-secret=abcdefghijklm123456789
```

7. Use the CredHub CLI to retrieve the credentials :
    - Retrieve the password for the admin user by running:

      ```
      credhub get -n /p-bosh/service-instance_GUID/admin_password
      ```

      In the output, the password appears under `value`. Record the password.
      For example:

      ```
      $ credhub get \
        -n /p-bosh/service-instance_70d30bb6-7f30-441a-a87c-05a5e4afff26/admin_
      password

        id: d6e5bd10-3b60-4a1a-9e01-c76da688b847
        name: /p-bosh/service-instance_70d30bb6-7f30-441a-a87c-05a5e4afff26/adm
      in_password
        type: password
        value: UMF2DXsqNPPlCNWMdVMcNv7RC3Wi10
        version_created_at: 2018-04-02T23:16:09Z
      ```

Identify Apps using a Service Instance

To identify which apps are using a specific service instance from the name of the BOSH deployment:

1. Take the deployment name and strip the `service-instance_` leaving you with the GUID.
2. Log in to CF as an admin.
3. Obtain a list of all service bindings by running the following:

```
cf curl /v2/service_instances/GUID/service_bindings
```

4. The output from the above curl gives you a list of `resources`, with each item referencing a service binding, which contains the `APP-URL`. To find the name, org, and space for the app, run the following:
    1. `cf curl APP-URL` and record the app name under `entity.name`.
    2. `cf curl SPACE-URL` to obtain the space, using the `entity.space_url` from the above

curl. Record the space name under `entity.name`.

3. `cf curl ORGANIZATION-URL` to obtain the org, using the `entity.organization_url` from the above curl. Record the organization name under `entity.name`.

Note: When running `cf curl` ensure that you query all pages, because the responses are limited to a certain number of bindings per page. The default is 50. To find the next page curl the value under `next_url`.

View BOSH Resource Saturation and Scaling

To view usage statistics for any service, do the following:

1. Run the following command:

```
bosh -d DEPLOYMENT-NAME vms --vitals
```

2. To view process-level information, run:

```
bosh -d DEPLOYMENT-NAME instances --ps
```

Monitor Quota Saturation and Service Instance Count

Quota saturation and total number of service instances are available through ODB metrics emitted to Loggregator. The metric names are shown below:

Metric Name	Description
`on-demand-broker/SERVICE-NAME-MARKETPLACE/quota_remaining`	global quota remaining for all instances across all plans
`on-demand-broker/SERVICE-NAME-MARKETPLACE/PLAN-NAME/quota_remaining`	quota remaining for a particular plan
`on-demand-broker/SERVICE-NAME-MARKETPLACE/total_instances`	total instances created across all plans
`on-demand-broker/SERVICE-NAME-MARKETPLACE/PLAN-NAME/total_instances`	total instances created for a given plan

Note: Quota metrics are not emitted if no quota has been set.

Reinstall a Tile

To reinstall a tile in the same environment where it was previously uninstalled:

1. Ensure that the previous tile was correctly uninstalled as follows:
    1. Log in as an admin by running:

```
cf login
```

    2. Confirm that the Marketplace does not list On-Demand Services SDK by running:

```
cf m
```

3. Log in to BOSH as an admin by running:

```
bosh log-in
```

4. Display your BOSH deployments to confirm that the output does not show the On-Demand Services SDK deployment by running:

```
bosh deployments
```

5. Run the "delete-all-service-instances" errand to delete every instance of the service.
6. Run the "deregister-broker" errand to delete the service broker.
7. Delete the service broker BOSH deployment by running:

```
bosh delete-deployment BROKER-DEPLOYMENT-NAME
```

8. Reinstall the tile.

Knowledge Base (Community)

Find the answer to your question and browse product discussions and solutions by searching the VMware Tanzu Knowledge Base.

File a Support Ticket

You can file a ticket with Support. Be sure to provide the error message from `cf service YOUR-SERVICE-INSTANCE`.

To expedite troubleshooting, provide your service broker logs and your service instance logs. If your `cf service YOUR-SERVICE-INSTANCE` output includes a `task-id`, provide the BOSH task output.

Create a pull request or raise an issue on the source for this page in GitHub

Service and Tile Developer Guide

Getting Started: ODB on a Local Development Environment

This topic describes how to create and manage an on-demand service broker (ODB) on a local development machine using Pivotal Cloud Foundry (PCF) Dev and BOSH Lite.

For more information about the components in this topic, see PCF Dev and BOSH-Lite.

Note: The examples in this topic are based on Kafka open source messaging. See the following sample code directories:

- Kafka example service
- Kafka example service adapter
- Kafka example app

Prerequisites

Before you set up and use ODB on your local machine, install and configure the following components:

- **BOSH Lite v9000.131.0 or later.** To install BOSH Lite, see Install in the BOSH Lite documentation.
- **PCF Dev file pcfdev-v0.19.1-rc.46.** To install PCF Dev, see Installing PCF Dev. Record the PCF Dev domain for later. The default is `local.pcfdev.io`.

Part 1: Set Up

This section details how to prepare BOSH Lite and set up the Kafka example service, the Kakfa example service adapter, and ODB.

Step 1: Prepare BOSH Lite

To prepare BOSH Lite, do the following:

1. Target your BOSH Lite installation.

```
bosh alias-env lite -e 192.168.50.4
```

2. Upload the BOSH Lite stemcell.

```
bosh -e lite upload-stemcell \
```

```
https://bosh.io/d/stemcells/bosh-warden-boshlite-ubuntu-trusty-go_agent?v=3262.
2
```

Step 2: Set Up the Kafka Example Service

To set up the Kafka example service, do the following:

1. Clone the Kafka example service into your workspace.

    ```
    git clone \
    https://github.com/pivotal-cf-experimental/kafka-example-service-release.git
    ```

2. In the `kafka-example-service-release` directory, create and upload the Kafka example service.

    ```
    cd kafka-example-service-release
    bosh create-release --name kafka-example-service
    ```

3. Upload the service to the BOSH director.

    ```
    bosh -e lite upload-release
    ```

Step 3: Set Up the Kafka Example Service Adapter

To set up the Kafka example service adapter, do the following:

1. Clone the Kafka example service adapter.

    ```
    git clone \
    https://github.com/pivotal-cf-experimental/kafka-example-service-adapter-releas
    e.git
    ```

2. Update service adapter dependencies.

    ```
    cd kafka-example-service-adapter-release
    git submodule update --init --recursive
    ```

3. Create the example service adapter.

    ```
    bosh create-release --name kafka-example-service-adapter
    ```

4. Upload the example service adapter to the BOSH director.

    ```
    bosh -e lite upload-release
    ```

Step 4: Set Up ODB

To set up ODB, do the following:

1. Download the on-demand service broker from Pivotal Network. To download, see Pivotal Cloud Foundry On Demand Services SDK.

2. Upload the `on-demand-service-broker` release.

```
bosh -e lite upload-release on-demand-service-broker-X.Y.Z.tgz
```

Where:

- `X.Y.Z` is the ODB release version.

  For example:

  ```
  $ bosh -e lite upload-release on-demand-service-broker-0.21.1.tgz
  ```

Part 2: Create

This section describes how to create a BOSH deployment and a service broker on PCF Dev.

Step 1: Create a BOSH Deployment

To create a BOSH Lite deployment, do the following:

1. Create a new directory in your workspace and a `cloud_config.yml` for the BOSH Lite Director. For example:

   ```
   vm_types:
   - name: container
     cloud_properties: {}

   networks:
   - name: kafka
     type: manual
     subnets:
     - range: 10.244.1.0/24
       gateway: 10.244.1.1
       az: lite
       cloud_properties: {}

   disk_types:
   - name: ten
     disk_size: 10_000
     cloud_properties: {}

   azs:
   - name: lite
     cloud_properties: {}

   compilation:
     workers: 2
     reuse_compilation_vms: true
     network: kafka
     az: lite
     cloud_properties: {}
   ```

2. Update the BOSH Lite cloud config using the deployment manifest.

   ```
   bosh -e lite update-cloud-config cloud_config.yml
   ```

3. Record the URL and UUID of your BOSH Lite director.

```
bosh environment
```

See the following example output:

```
$ bosh environment
Config
             /Users/pivotal/.bosh_config



Director
  Name       Bosh Lite Director
  URL        https://192.168.50.4:25555
  Version    1.3215.0 (00000000)
  User       admin
  UUID       17a45148-1d00-43bc-af28-9882e5a6535a
  CPI        warden_cpi
  dns        disabled
  compiled_package_cache enabled (provider: local)
  snapshots  disabled
```

4. Create a BOSH Lite deployment manifest in a file called `deployment_manifest.yml` using the following as a base:

```
name: kafka-on-demand-broker

director_uuid: BOSH-LITE-UUID

releases:
- name: &broker-release on-demand-service-broker
  version: latest
- name: &service-adapter-release kafka-example-service-adapter
  version: latest
- name: &service-release kafka-example-service
  version: latest

stemcells:
- alias: trusty
  os: ubuntu-trusty
  version: STEMCELL-VERSION

instance_groups:
- name: broker
  instances: 1
  vm_type: container
  persistent_disk_type: ten
  stemcell: trusty
  azs: [lite]
  networks:
  - name: kafka
  jobs:
  - name: kafka-service-adapter
    release: *service-adapter-release
  - name: admin_tools
    release: *service-release
  - name: broker
    release: *broker-release
```

```
    properties:
      port: 8080
      username: broker    # or replace with your own
      password: password  # or replace with your own
      disable_ssl_cert_verification: true
      bosh:
        url: BOSH-LITE-URL
        authentication:
          basic:
            username: admin
            password: admin
      cf:
        url: https://api.PCF-DEV-DOMAIN
        authentication:
          url: https://uaa.PCF-DEV-DOMAIN
          user_credentials:
            username: admin
            password: admin
      service_adapter:
        path: /var/vcap/packages/odb-service-adapter/bin/service-adapter
      service_deployment:
        releases:
        - name: *service-release
          version: SERVICE-RELEASE-VERSION
          jobs: [kafka_server, zookeeper_server]
        stemcells:
          - os: ubuntu-trusty
            version: STEMCELL-VERSION
      service_catalog:
        id: D94A086D-203D-4966-A6F1-60A9E2300F72
        service_name: kafka-service-with-odb
        service_description: Kafka Service
        bindable: true
        plan_updatable: true
        tags: [kafka]
        plans:
        - name: small
          plan_id: 11789210-D743-4C65-9D38-C80B29F4D9C8
          description: A Kafka deployment with a single instance of each job an
d persistent disk
          instance_groups:
          - name: kafka_server
            vm_type: container
            instances: 1
            persistent_disk_type: ten
            azs: [lite]
            networks: [kafka]
          - name: zookeeper_server
            vm_type: container
            instances: 1
            persistent_disk_type: ten
            azs: [lite]
            networks: [kafka]
          properties:
            auto_create_topics: true
            default_replication_factor: 1

update:
  canaries: 1
```

```
  canary_watch_time: 30000-180000
  update_watch_time: 30000-180000
  max_in_flight: 4
```

Where:

- `BOSH-LITE-UUID` is the `UUID` value you recorded in the BOSH environment step above.
- `BOSH-LITE-URL` is the `URL` value you recorded in the BOSH environment step above.
- `PCF-DEV-DOMAIN` is the PCF Dev domain you recorded in the Prerequisites above.

5. Deploy the broker.

```
bosh -e lite -d kafka-on-demand-broker deployment_manifest.yml
```

6. Record the IP address of the deployed broker.

```
bosh -e lite -d kafka-on-demand-broker instances
```

See the following example output:

```
$ Using environment 'lite' as user 'admin' (openid, bosh.admin)

Task 54727. Done

Deployment 'redis-on-demand-broker-dev2'

Instance                                      Process State  AZ  IPs
broker/84294753-84b9-4be1-a338-37c1f3e71919   running        z1  10.244.1.2

1 instances

Succeeded
```

Step 2: Create a Service Broker on PCF Dev

To create a service broker on PCF Dev, do the following:

1. Create a service broker on PCF Dev and enable access to its service offering.

```
cf create-service-broker kafka-broker USERNAME PASSWORD http://BROKER-IP:8080
```

Where:

- `USERNAME` and `PASSWORD` are the broker's credentials set under `properties` in the broker job.
- `BROKER-IP` is the value obtained in the step above. See the last step in Create a BOSH Deployment.

For example:

```
$ cf create-service-broker kafka-broker broker password http://10.244.1.2:8080
```

2. Enable access to the broker's service plans.

```
cf enable-service-access kafka-service-with-odb
```

3. View the broker-offered services in the Marketplace.

```
cf marketplace
```

See the following example output:

```
Getting services from Marketplace in org pcfdev-org / space pcfdev-space as adm
in...
OK

service                   plans        description
kafka-service-with-odb    small        Kafka Service
p-mysql                   512mb, 1gb   MySQL databases on demand
p-rabbitmq                standard     RabbitMQ is a robust and scalable high-pe
rformance multi-protocol messaging broker.
p-redis                   shared-vm    Redis service to provide a key-value stor
e
```

4. Create a service instance using the Kafka on-demand broker.

```
cf create-service kafka-service-with-odb small k1
```

Part 3: Verify and Use

To verify and use your on-demand service, do the following:

Step 1: Verify Your BOSH Deployment and On-Demand Service

1. Check the status of your service.

```
cf service k1
```

See the state change from `create in progress` to `create succeeded`.

2. Verify that the on-demand service is provisioned in the BOSH deployment

```
bosh -e lite deployments
```

See the following example output:

```
Name                                                    Release(s)
                  Stemcell(s)                                        Cloud Confi
g
kafka-on-demand-broker                                  kafka-example-service-a
dapter/0+dev.2    bosh-warden-boshlite-ubuntu-trusty-go\_agent/3262.2   latest
                                                        on-demand-service-broke
r/0.2.0+dev.1
service-instance_2715262c-8564-4cd9-b629-0ae99e6aa4b9   kafka-example-service/0
+dev.2            bosh-warden-boshlite-ubuntu-trusty-go\_agent/3262.2   latest
```

This example shows that the service instance is provisioned and the service releases are

specified in the ODB deployment manifest.

Step 2: Use Your On-Demand Service

To use the service instance that you created, do the following:

1. Clone the Kafka example app.

```
git clone https://github.com/pivotal-cf-experimental/kafka-example-app.git
```

2. Push the app.

```
cd kafka-example-app
cf push --no-start
```

3. Bind the app to your service instance.

```
cf bind-service kafka-example-app k1
```

4. Start the app.

```
cf start kafka-example-app
```

Step 3: Read and Write to Your Service Instance

Now the app runs at `https:/kafka-example-app.PCF-DEV-DOMAIN`. You can use it to read and write to your on-demand Kafka service instance.

For example:

- To write data, run the following.

```
curl -XPOST http://kafka-example-app.PCF-DEV-DOMAIN/queues/my-queue -d SOME-DAT
A
```

- To read data, run the following.

```
curl http://kafka-example-app.PCF-DEV-DOMAIN/queues/my-queue
```

Create a pull request or raise an issue on the source for this page in GitHub

Creating a Service Release

This topic provides information for service authors about how to create a service release for an on-demand service tile.

Service Author Deliverables

Service authors provide the following:

- The Service Release. For more information, see Overview below.
- The Service Adapter. For more information, see Creating a Service Adapter.

- Documentation for the operator to configure plan definitions for the Service Adapter.
- Documentation for the operator to backup and restore service instances.

Overview

A service release is a BOSH release of the service that you want to create on-demand instances of. The on-demand broker (ODB) deploys this release at instance creation time, once for each service instance.

To create a service release, Pivotal recommends that you do the following:

1. Package an Initial Service Release
2. Refine your release using the information in the following sections:
    - Use Job Links
    - Service Instance Lifecycle Errands
3. Package the Final Service Release

Package an Initial Service Release

Package an initial version of the BOSH release for your service to test whether it deploys successfully.

To do so, create and manually deploy a BOSH release for the service by following the instructions up to and including *Step 6: Create a Dev Release* in Creating a Release in the BOSH Documentation.

Note: Pivotal recommends that you create sample manifests that deploy the service release(s). This helps you to write the `generate-manifest` component of the service adapter. For help writing a sample manifest, see Deployment Config in the BOSH documentation.

Example service releases:

- redis-example-service-release
- kafka-example-service-release

Use Job Links

When there are multiple jobs in the manifest that need to communicate over the network, Pivotal recommends that you use BOSH' s job link feature instead of using static IP addresses. IP addresses must be different for each service instance. When you use job links, BOSH inserts IP addresses or internal DNS names when templating the job configuration so you do not have to do it manually.

Job links are defined in the service release and configured in the manifest. For how to use job links, see Links in the BOSH documentation.

For an example, see the kafka-example-service-release on GitHub. The example uses implicit job links to get the IP addresses of the brokers and the zookeeper.

Service Instance Lifecycle Errands

Note: This feature requires BOSH Director v261 or later.

Service instance lifecycle errands allow additional short-lived jobs to run as part of service instance deployment. ODB uses these errands to manage an instance lifecycle. A deployment is only considered successful if the deployment and all lifecycle errands complete successfully.

ODB supports the following service instance lifecycle errands:

- **Post-deploy**: These errands run after creating or updating a service instance. For example, running a health check to ensure the service instance is functioning. To see the workflow for post deploy errands, see Create or Update Service Instance with Post-Deploy Errands.

- **Pre-delete**: These errands run before the deletion of a service instance. For example, cleaning up data before a service shutdown. To see the workflow for pre-delete errands, see Delete a Service Instance with Pre-Delete Errand.

For information for operators about how to enable these errands in the manifest, see Enable Service Instance Lifecycle Errands.

Include Service Instance Lifecycle Errands

Lifecycle errands are defined in the service release. The service adapter or operator can configure these errands when generating a manifest.

To include lifecycle errands in your service release, do the following:

1. Decide what errands your on-demand service needs. For example, you could create a health check post-deploy errand using the criteria that you used to test the initial release.
2. Write code to run each lifecycle errand and define them as jobs in the service release. For how to do so, see the Using Errands in the BOSH documentation.

For an example implementation of a health check post-deploy errand, see the redis-example-service-release on GitHub.

Note: When using the service adapter `generate-manifest` command, you must validate that any lifecycle errands configured in the `plan` parameter exist in the service release and are included in the service manifest as jobs. For more information about the generate manifest command, see generate-manifest.

Colocated Errands

Note: This feature requires BOSH Director v263 or later.

Colocated errands run on an existing service instance group, avoiding additional resource allocation. Both `post-deploy` and `pre-delete` errands can be run as colocated errands.

To enable a new colocated errand, add the errand to the list of jobs of an instance group.

Package the Final Service Release

To package the final release, follow the instructions in Create a Final Release in the BOSH documentation.

The tile author packages this release into the tile. For direct BOSH deployments, an operator uploads this release to the BOSH Director.

Create a pull request or raise an issue on the source for this page in GitHub

Creating a Service Adapter

This topic provides information for service authors about how to create a service adapter for an on-demand service tile. For more information about service author responsibilities, see Service Author Deliverables.

About Service Adapters

A service adapter is an executable invoked by the on-demand broker (ODB). Pivotal has published an SDK for teams writing their service adapters in golang. For more information, see On-Demand Services SDK Golang SDK below.

Subcommands in the Adapter Interface

A service adapter is expected to respond to the subcommands below. For more information about the parameters and expected output from these subcommands, see Service Adapter Interface Reference.

- `generate-manifest`: Generate a BOSH manifest for your service instance deployment and output to stdout a JSON document containing the manifest as YAML, BOSH secrets, and BOSH Configs, given information about the:
    - BOSH Director (stemcells, release names)
    - service instance (ID, request parameters, plan properties, IaaS resources)
    - previous manifest, if this is an upgrade deployment

    Note: ODB requires `generate-manifest` to be idempotent. Given the same arguments when a previous manifest is supplied—which happens during a deployment update—the command should always output the same BOSH manifest.

    For more information about this subcommand, see generate-manifest.

- `dashboard-url`: Generate an optional URL of a web-based management UI for the service instance. For more information about this subcommand, see dashboard-url.

- `create-binding`: Create credentials for the service instance, printing them to stdout as JSON.

These should be unique, if possible. For more information about this subcommand, see create-binding.

- `delete-binding`: Invalidate the created credentials, if possible. Some services (e.g. Redis) are single-user, and this endpoint does nothing. For more information about this subcommand, see delete-binding.

- `generate-plan-schemas`: Generate a JSON schema to validate service-specific configuration parameters. For more information about this subcommand, see generate-plan-schemas.

Store Secrets on BOSH CredHub

The service adapter can generate secrets and use ODB as a proxy to the BOSH CredHub Config server, instead of writing these secrets in plaintext in the manifest. To do this, use ODB-managed secrets.

The following sections provide information about how to use ODB-managed secrets to store, persist, and modify secrets in BOSH CredHub:

- About ODB-Managed Secrets
- Migrate from Plaintext Secrets to ODB-Managed Secrets
- Persist Credentials Across Updates
- Modify ODB-Managed Secrets

About ODB-Managed Secrets

To use ODB-managed secrets, the service adapter must do the following for the `generate-manifest` output:

1. Generate a manifest that uses the ODB-managed secrets placeholder `((odb_secret:SECRET-NAME))` for the secret that you want to store in BOSH CredHub.
2. Output the secret as part of the `secrets` map.

For example:

```
{
  "manifest": "password: ((odb_secret:SECRET-NAME))",
  "secrets": {
    "SECRET-NAME":"SOME-RANDOM-PASSWORD"
  }
}
```

When you use ODB-managed secrets, ODB does the following during provision:

1. Generates a BOSH CredHub name for each secret in the `secrets` map in the format `/odb/SERVICE-OFFERING-ID/SERVICE-INSTANCE-ID/SECRET-NAME`.
2. Saves the value of the secret in the BOSH CredHub using the generated name.
3. Replaces all occurrences of `((odb_secret:SECRET-NAME))` with the generated BOSH

CredHub name.

4. Deploys the updated manifest.

Migrate from Plaintext Secrets to ODB-Managed Secrets

You can use the service adapter to migrate from plaintext secrets in the manifest to ODB-managed secrets that are stored in BOSH CredHub. When the `generate-manifest` subcommand is provided with a previous manifest, the service adapter copies secrets from the previous deployment to the new manifest.

To migrate from plaintext secrets to ODB-managed secrets, write code in your service adapter that does the following:

Note: Secrets already stored in BOSH CredHub do not need placeholders. This is because ODB ignores BOSH CredHub names during `generate-manifest`.

- Detects whether a secret is a plaintext secret.
- Replaces each plaintext secret from the previous manifest with an ODB-managed secrets placeholder `((odb_secret:SECRET-NAME))` in the new manifest.
- Returns the value of the secrets in the `secrets` map. For more information about the `secrets` map, see About ODB-Managed Secrets above.
    - Each placeholder in the manifest a corresponding entry in the `secrets` map.
    - Each key in the `secrets` map at least one corresponding placeholder in the manifest.
- Only returns secrets in the `secrets` map when the value of the secret is set for the first time, or if the value is changed. For example, this might be when:
    - A new service is created.
    - You migrate plaintext secrets into BOSH CredHub.
    - You want to change the value for previously set secrets.

For example:

```
import(
  "github.com/pivotal-cf/on-demand-services-sdk/serviceadapter"
)

func extractSecret(oldValue, secretName string, secretsMap map[string]string, newValue
 string) {
  if !( strings.HasPrefix(redisPassword, "((") && strings.HasSuffix(redisPassword, "))
") ) {
    // This is a plaintext secret
    // Add the value to the secrets map,
    secretsMap[secretName] = oldValue
    // and return its placeholder to use in the manifest.
    newValue = fmt.Sprintf( "((%s:%s))", serviceadapter.ODBSecretPrefix, secretName)
    return newValue, secretsMap
  } else {
    // else: this secret could be one of the following:
    // - a custom CredHub name
```

```
    // - a reference to the BOSH generated variables block
    // - a CredHub reference to a secret already managed by the ODB
    // In all cases, the ODB does not need to send the secret to CredHub, so it
    // should not be included in the secrets map.
    return oldValue
  }
}
```

Persist Secrets across Updates

When dealing with properties that need to persist across updates, the service adapter must extract the existing name for any ODB-managed secrets from the previous manifest.

The following manifest snippet shows an ODB-managed secret with a BOSH CredHub name:

```
name: the-deployment
...
properties:
password: ((/odb/SERVICE-GUID/SERVICE-INSTANCE-GUID/SECRET-NAME))
```

If the previous manifest contains BOSH CredHub names for secrets, the `generate-manifest` command must not replace `properties.password` with the placeholder `((odb_secret:SECRET-NAME))`.

For more information about using the value during a bind, see create-binding.

Modify ODB-Managed Secrets

WARNING: Pivotal discourages modifying the value of secrets without changing the secret name. If the BOSH deploy task fails during update or upgrade, ODB-managed secrets might be left in an inconsistent state. For more information, see Inconsistent Secrets after a Failed Update below.

When updating or upgrading a service instance, operators might need to modify the value of an ODB-managed secret. These secrets are passed to the service adapter from the following:

- Plan properties in the on-demand broker manifest
- Adapter secrets given in the adapter config
- Configuration parameters in the `cf update-service` command

To regenerate the manifest with modified secrets, write code in your service adapter that does the following:

1. Replaces the property in the manifest with an ODB-managed secrets placeholder that uses a new secret name.
2. Uses the `GenerateManifest` method to return the new secret in the `secrets` map.

Detect When Secrets Are Modified

The service adapter must only insert the ODB-managed secrets placeholder if a secret has been modified. This is because ODB requires that the `GenerateManifest` method is idempotent. When the

service adapter generates a new manifest after a deployment update, it must be the same as the previous manifest when `GenerateManifest` is given the same input.

ODB provides all the currently deployed secrets to the `GenerateManifest` method using the `previousSecrets` argument. For more information about the input to the `previousSecrets` argument, see PREVIOUS-SECRETS-JSON.

To detect whether a secret has been modified, write code in your service adapter that does the following:

1. Compares the previous value of the secret to the new value.
2. **If the secret has changed:**
    1. Inserts the ODB-managed secrets placeholder.
    2. Adds the value to the `secrets` map.

    If the secret has remained the same:
    1. Inserts the BOSH CredHub name from the previous manifest.

For example code that does the above, see the Redis Example Adapter.

Inconsistent Secrets after a Failed Update

If you modify the value of a secret without providing a new secret name, ODB-managed secrets can be left in an inconsistent state if the update or upgrade of a BOSH deployment fails. This is because ODB updates the secrets in BOSH CredHub before updating the deployment.

The failed deployment might contain a mixture of old and new secrets depending on the stage that the deployment failed. When an operator attempts to troubleshoot this scenario by manually re-deploying the previous manifest, this manifest contains BOSH CredHub names that refer to the new secret values. This can cause errors with bindings.

Pivotal recommends that you avoid modifying secrets without using new names for new versions of secrets.

Binding Credentials

Ensure binding credentials for a service instance share a namespace and are unique, if possible.

For MySQL, two bindings could include different username/password pairs, but share the same MySQL database tables and data. The first step is to determine which credentials are best to supply in the context of your service. Pivotal recommends that users are identified statelessly from the binding ID. The simplest way to do this is to name the user after the binding ID.

You can take one of three approaches to credentials for a service binding:

- Static Credentials
- Credentials Unique to Each Binding
- Use an Agent

Static Credentials

In this case, the same credentials are used for all bindings. One option is to define these credentials

in the service instance manifest.

This scenario makes sense for services that use the same credentials for all bindings, such as Redis.

For example:

```
properties:
  redis:
    password: PASSWORD
```

Credentials Unique to Each Binding

In this case, when the adapter `generate-manifest` subcommand is invoked, it generates random admin credentials and returns them as part of the service instance manifest. When the `create-binding` subcommand is invoked, the adapter can use the admin credentials from the manifest to create unique credentials for the binding. Subsequent `create-binding` calls create new credentials.

This option makes sense for services whose binding creation resembles user creation, such as MySQL or RabbitMQ. For example, in MySQL the admin user can be used to create a new user and database for the binding:

```
properties:
  admin_password: ADMIN-PASSWORD
```

Use an Agent

In this case, the author defines an agent responsible for handling the creation of credentials unique to each binding. The agent must be added as a BOSH release in the service manifest. Moreover, the service and agent jobs should be colocated in the same instance group.

This option is useful for services where the adapter cannot, or tends not, to directly call out to the service instance and instead delegates responsibility for setting up new credentials to an agent.

For example:

```
releases:
  - name: service-release
    version: 1.5.7
  - name: credentials-agent-release
    version: 4.2.0

instance_groups:
  - name: service-group
    jobs:
      - name: service-job
        release: service-release
      - name: credentials-agent-job
        release: credentials-agent-release
```

Enable ODB to Obtain BOSH DNS Addresses

Note: This feature requires v266.3 or later of the BOSH Director. This is available in Ops Manager v2.2 and later.

You can configure ODB to provide BOSH DNS addresses for service instances to the service adapter `create-binding` and `delete-binding` calls. This is useful when the binding for a service instance contains, or relies on, BOSH DNS addresses for that deployment. For more information about how DNS addresses are passed to the `create-binding` and `delete-binding` calls, see DNS-ADDRESSES-JSON.

To enable ODB to provide service instance DNS addresses to the `create-binding` and `delete-binding` calls, do the following:

1. Provide a link from the service instance' s BOSH release. Choose any job in the service release and add the link to its `spec` file.

   For example:

   ```
   name: redis-server-job
   ...
   provides:
     - name: example-link-1
       type: example-type
   ```

   For an example `spec` file, see the Redis Example Service Release.

2. Write code in the service adapter that shares the link you provided above in the BOSH manifest generated for your service instance deployment. Share the link in the same job that you added the link to in step 1. Include the link in all instance groups that require a DNS address at binding time.

   For example:

   ```
   instance_groups:
   - name: leader-node
     jobs:
     - name: redis-server-job
       release: redis-cluster-release
       provides:                          # add this section
         example-link-1: {shared: true}
       ...
   ```

Use Generic BOSH Configs with Service Instances

The service adapter can generate generic BOSH configs and use ODB to apply them to the BOSH Director before deployment of the service instance. This enables the service author to provide service instance-specific BOSH configs which exist only for the lifetime of the service instance.

See the BOSH documentation for more details about generic BOSH configs.

To return a BOSH config fragment specific to a service instance manifest, it must be included in the response from the `generate-manifest` command, as in the example below.

```
{
  "manifest": "
```

```
    name: MY-SERVICE-INSTANCE
    instance_groups:
    - vm_type: MY-SERVICE-INSTANCE-small",
  "configs": {
    "cloud":"
       vm_types:
       - name: MY-SERVICE-INSTANCE-small
         cloud_properties:
           cpu: 1"
  }
}
```

Where `MY-SERVICE-INSTANCE` is your service instance.

ODB takes the configs in the output and for each entry creates a generic BOSH config on the director, using the map key as the config type and using the service instance deployment name as the config name. If the example above were applied using the BOSH CLI, it would look similar to this:

```
$ echo <content of configs["cloud"]> > config.yml
$ bosh update-config --type cloud --name MY-SERVICE-INSTANCE config.yml
```

The valid config types are: cloud, cpi and runtime.

The configs are scoped for the BOSH team client ODB is deployed with and they are applied to subsequently deployed service instances.

Note: Pivotal recommends that service adapters also namespace their configuration to avoid depending on a config value, which might be deleted in the future when the associated service instance is removed.

On updates, ODB passes the BOSH configs previously set for the instance to the service adapter on `generate-manifest`. When the service adapter returns a new value for an existing type, the configuration is overwritten. When no value is returned for an existing type, that type remains as is. When new types are passed, those are set.

When the service instance is deleted, all the associated BOSH configs are also be deleted.

Handle Errors

If a subcommand fails, the adapter must return a non-zero exit status and, optionally, print to stdout or stderr.

When a subcommand exits with an unrecognized exit code anything printed to stdout is returned to the CF CLI user.

Both the stdout and stderr streams are printed in the broker log for the operator. For that reason, Pivotal recommends not printing the manifest or other sensitive details to stdout or stderr, because ODB does no validation on this output.

For an example implementation, see Kafka example service adapter on GitHub.

Package a Service Adapter

Package the service adapter as a BOSH release. The operator should colocate the service adapter release with the ODB release in a BOSH manifest to place the adapter executable on the same VM as the ODB server. As a result, the adapter BOSH job's `monit` file should have no processes defined.

See the following example service adapter releases:

- kafka-example-service-adapter-release
- redis-example-service-adapter-release

For more information about how to create a BOSH release, see Creating a Release in the BOSH documentation.

On-Demand Services Golang SDK

Pivotal has published an SDK for teams writing their service adapters in golang. It covers command line invocation handling, parameter parsing, response serialization, and error handling so the adapter authors can focus on the service-specific logic in the adapter. For more information about the Golang SDK, see the on demand service SDK repository on GitHub.

The SDK supports properties in two levels for the generated BOSH manifest, manifest global and job level. Global properties are deprecated in BOSH, in favor of job level properties and job links.

For an example of property generation, see the Kafka example service adapter.

Use the SDK

Perform the following steps to use the SDK:

1. Install the SDK by running the following `go get` command:

   ```
   go get github.com/pivotal-cf/on-demand-services-sdk
   ```

   Use the same version of the SDK as your ODB release. For example, if you are using v0.8.0 of the ODB BOSH release, you should check out the v0.8.0 tag of the SDK.

2. In the main function for the service adapter, call the `HandleCLI` function:

   ```
   package main

   import (
       "log"
       "os"

       "URL-FOR-SERVICE-ADAPTER-REPOSITORY"
       "github.com/pivotal-cf/on-demand-services-sdk/serviceadapter"
   )

   func main() {
     logger := log.New(os.Stderr, "[SERVICE-ADAPTER-NAME] ", log.LstdFlags)
     manifestGenerator := adapter.ManifestGenerator{}
     binder := adapter.Binder{}
     dashboardUrlGenerator := adapter.DashboardUrlGenerator{}
       handler := serviceadapter.CommandLineHandler{
       ManifestGenerator:     manifestGenerator,
   ```

```
    Binder:                binder,
    DashboardURLGenerator: &adapter.DashboardUrlGenerator{},
    SchemaGenerator:       adapter.SchemaGenerator{},
    }
  serviceadapter.HandleCLI(os.Args, handler)
}
```

Where:

- `URL-FOR-SERVICE-ADAPTER-REPOSITORY` is the repository containing your service adapter, for example `github.com/bar-org/foo-service-adapter/adapter`.
- `SERVICE-ADAPTER-NAME` is the name of the service adapter, for example `foo-service-adapter`.

Note: The HandleCommandLineInvocation function is deprecated, but to see its functionality, see Usage.

Interfaces

The `HandleCLI` function accepts structs that implement the interfaces below. For more information about the corresponding adapter interfaces, see Subcommands in the Adapter Interface.

```
type CommandLineHandler struct {
    ManifestGenerator     ManifestGenerator
    Binder                Binder
    DashboardURLGenerator DashboardUrlGenerator
    SchemaGenerator       SchemaGenerator
}
```

Service adapters provide the following to the `CommandLineHandler`:

- A `ManifestGenerator`, required for all service adapters:

```
type ManifestGenerator interface {
    GenerateManifest(params GenerateManifestParams) (GenerateManifestOutput, er
ror)
}

type GenerateManifestParams struct {
    ServiceDeployment ServiceDeployment
    Plan              Plan
    RequestParams     RequestParameters
    PreviousManifest  *bosh.BoshManifest
    PreviousPlan      *Plan
    PreviousSecrets   ManifestSecrets
    PreviousConfigs   BOSHConfigs
}

type GenerateManifestOutput struct {
    Manifest          bosh.BoshManifest `json:"manifest"`
    ODBManagedSecrets ODBManagedSecrets `json:"secrets"`
    Configs           BOSHConfigs       `json:"configs"`
}
```

- A `Binder`, required for most service adapters:

```
type Binder interface {
    CreateBinding(params CreateBindingParams) (Binding, error)
    DeleteBinding(params DeleteBindingParams) error
}

type CreateBindingParams struct {
    BindingID          string
    DeploymentTopology bosh.BoshVMs
    Manifest           bosh.BoshManifest
    RequestParams      RequestParameters
    Secrets            ManifestSecrets
    DNSAddresses       DNSAddresses
}

type DeleteBindingParams struct {
    BindingID          string
    DeploymentTopology bosh.BoshVMs
    Manifest           bosh.BoshManifest
    RequestParams      RequestParameters
    Secrets            ManifestSecrets
    DNSAddresses       DNSAddresses
}
```

- A `DashboardUrlGenerator`, optional:

```
type DashboardUrlGenerator interface {
    DashboardUrl(params DashboardUrlParams) (DashboardUrl, error)
}

type DashboardUrlParams struct {
    InstanceID string
    Plan       Plan
    Manifest   bosh.BoshManifest
}
```

- A `SchemaGenerator`, optional

```
type SchemaGenerator interface {
    GeneratePlanSchema(params GeneratePlanSchemaParams) (PlanSchema, error)
}

type GeneratePlanSchemaParams struct {
    Plan Plan
}
```

Helpers

The helper function `GenerateInstanceGroupsWithNoProperties` can generate the instance groups for the BOSH manifest from the arguments passed to the adapter.

One of the inputs for this function is `deploymentInstanceGroupsToJobs`, where instance groups are mapped to jobs for the deployment. The service author must provide this mapping. The helper function does not address job level properties for the generated instance groups; the service author must provide these properties. For an example implementation, see the job mapping in the Kafka example adapter on GitHub.

The SDK provides the methods `ArbitraryContext` and `Platform`. These are used to extract the `context` property from the request parameters and the `platform` property from within the `context`.

The context in the response is a feature of Open Service Broker API (OSBAPI) v2.13 specifications and is used to pass through information about the environment in which the platform or app is executing. See the OSBAPI v2.13 specification on GitHub for more information. If the platform does not provide a `context`, the SDK returns empty values.

Error Handling

Any error returned by the interface functions is considered to be for the Cloud Foundry CLI user and is printed to stdout.

The adapter code is responsible for performing any error logging to stderr that the authors think is relevant for the operator logs.

There are three specialized errors for the `CreateBinding` function, which allow the adapter to exit with the appropriate code:

```
serviceadapter.NewBindingAlreadyExistsError()
serviceadapter.NewBindingNotFoundError()
serviceadapter.NewAppGuidNotProvidedError()
```

For more complete code examples, see the Kafka example service adapter on GitHub and the Redis example service adapter on GitHub.

BOSH Features

Service authors can enable configuration of BOSH Features in their service adapters.

The SDK provides the `BoshFeatures` struct below, with the option to add extra features using the `ExtraFeatures` map:

```
type BoshFeatures struct {
    UseDNSAddresses      *bool                  `yaml:"use_dns_addresses,omitempty"`
    RandomizeAZPlacement *bool                  `yaml:"randomize_az_placement,omitempt
y"`
    UseShortDNSAddresses *bool                  `yaml:"use_short_dns_addresses,omitemp
ty"`
    ExtraFeatures        map[string]interface{} `yaml:"extra_features,inline"`
}
```

For an example implementation, see the Redis example service adapter in GitHub.

For more information about BOSH Features, see the BOSH documentation.

Create a pull request or raise an issue on the source for this page in GitHub

Creating an On-Demand Service Tile

This topic describes how to build an on-demand service tile using the Tile Generator. For an example tile, see the example-kafka-on-demand-tile in GitHub. For a list of available manifest properties for the broker, see the broker job spec in GitHub.

Requirements

To build an on-demand tile you need the following releases:

- **On Demand Service Broker (ODB)**—Download ODB from Pivotal Network.
- **Your service adapter**—Get this from the service author.
- **Your service release**—Get this from the release author.

About Networks

When using the ODB in a tile with Ops Manager v2.0 and earlier, you need at least two private networks:

- A network where Ops Manager deploys the on-demand broker VM
- A different network where the on-demand broker deploys service instance VMs

The network for service instances should be flagged as a Service Network in Ops Manager.

Note: For Ops Manager v2.1 and later, you do not need separate networks for the on-demand broker and service instances. However, Pivotal recommends that you have at least two networks as described above.

Build a Tile for an On-Demand Service

There are several methods you can use to build a tile. This topic describes how to build a tile using the Tile Generator.

To use the Tile Generator to build a tile for an on-demand service, do the following:

1. Generate a `tile.yml` file by doing steps 1 – 4 of How to Use.
2. Add accessors, on-demand broker lifecycle errands, and optional features to the `tile.yml` file generated in step 1. This provides configuration for the ODB and additional configuration options for operators to select in Ops Manager.

   For more information about what to add to the `tile.yml`, see the following sections below:

   - Add Accessors
   - Add On-Demand Broker Lifecycle Errands
   - (Optional) Allow Public IP Addresses for On-Demand Service Instance Groups
   - (Optional) Enable Floating Stemcells
   - (Optional) Allow Secure Binding

3. Build your tile by running the following command:

```
tile build
```

Add Accessors

The ODB requires tiles to be configured with certain information. You must add accessors to the `tile.yml` file to provide values that operators cannot configure in Ops Manager.

Add the following accessors to your `tile.yml` file:

Note: The accessors in this section are mandatory. For other accessors, see Ops Manager Provided Snippets.

director

Ops Manager uses these accessors to get values relating to the BOSH Director installation. For the on-demand broker to interact with BOSH Director, on-demand service tiles must be configured with credentials for managing BOSH deployments.

The following table lists the accessors you must add:

Accessor	Description
$director.hostname	The director's hostname or IP address
$director.ca_public_key	The director's root ca certificate. Related: Configure SSL Certificates.

For example:

```
bosh:
  url: https://(( $director.hostname )):25555
  root_ca_cert: (( $director.ca_public_key ))
```

To see this example in context, see the example-kafka-on-demand-tile.

self

Ops Manager uses these accessors to get values that have been assigned to the tile after installation. To enable `$self` accessors, set `service_broker: true` at the top level of your `tile.yml` file.

Note: Setting `service_broker: true` causes the BOSH Director to redeploy when installing or uninstalling the tile.

The following table lists the accessors you must add:

Accessor	Description
$self.uaa_client_name	UAA client name that can authenticate with the BOSH Director
$self.uaa_client_secret	UAA client secret that can authenticate with the BOSH Director

Accessor	Description
$self.stemcell_version	The stemcell that the service deployment uses
$self.service_network	Service network configured for the on-demand instances

The service network has to be created manually. Create a subnet on AWS and then add it to the director. In the BOSH Director tile, under Create Networks > ADD network > fill in the subnet/vpc details.

For example:

```
bosh:
  authentication:
    uaa:
      url: https://(( $director.hostname )):8443
      client_id: (( $self.uaa_client_name ))
      client_secret: (( $self.uaa_client_secret ))
```

To see this example in context, see the example-kafka-on-demand-tile.

(Optional) cf

Ops Manager uses these accessors to get values from the Pivotal Application Service (PAS) tile. If you want to use PAS, add these accessors to your `tile.yml` file.

The following table lists the accessors you must add to use PAS:

Accessor	Description
..cf.ha_proxy.skip_cert_verify.value	Flag to skip SSL certificate verification for connections to the CF API
..cf.cloud_controller.apps_domain.value	The application domain configured in the CF installation
..cf.cloud_controller.system_domain.value	The system domain configured in the CF installation
..cf.uaa.system_services_credentials.identity	Username of a CF user in the cloud_controller.admin group, to be used by services
..cf.uaa.system_services_credentials.password	Password of a CF user in the cloud_controller.admin group, to be used by services

For example:

```
disable_ssl_cert_verification: (( ..cf.ha_proxy.skip_cert_verify.value ))
cf:
  url: https://api.(( ..cf.cloud_controller.system_domain.value ))
  authentication:
    url: https://uaa.(( ..cf.cloud_controller.system_domain.value ))
    user_credentials:
      username: (( ..cf.uaa.system_services_credentials.identity ))
      password: (( ..cf.uaa.system_services_credentials.password ))
```

To see this example in context, see the example-kafka-on-demand-tile.

Add On-Demand Broker Lifecycle Errands

The example-kafka-on-demand-tile example shows how the errands in the on-demand broker release can be used.

Pivotal recommends that you add the errands below to your tile. The errands should be specified in the following order:

Post-deploy:

- `register-broker`
- `upgrade-all-service-instances`

Pre-delete:

- `delete-all-service-instances-and-deregister-broker`

For more information about these errands, see Broker and Service Management.

Upgrade All Service Instances Errand

The `upgrade-all-service-instances` errand can be configured with two parameters:

- The number of simultaneous upgrades
- The number of canary instances

For more information about these parameters, see Upgrade All Service Instances.

The example example-kafka-on-demand-tile shows how to create a tab with fields to configure the parameters for this errand. The example tile has constraints to ensure the number of simultaneous upgrades is greater than one and the number of canaries is greater than zero.

(Optional) Allow Public IP Addresses for On-Demand Service Instance Groups

Ops Manager provides a VM extension called `public_ip` in the BOSH Director's cloud config. Use this feature to give Ops Manager operators the option to assign a public IP address to instance groups. This IP is only used for outgoing traffic to the internet from VMs with the `public_ip` extension. All internal traffic / incoming connections need to go over the private IP.

To allow operators to a assign public IP addresses to on-demand service instance groups, update your `tile.yml` file as follows:

1. Add the following to the `form_types` section:

   For example:

   ```
   form_types:
   - name: example_form
     property_inputs:
     - reference: .broker.example_vm_extensions
       label: VM options
       description: List of VM options for Service Instances
   ```

2. Add the following to the `job_types` section:

For example:

```
  job_types:
  - name: broker
    templates:
    - name: broker
      release: on-demand-service-broker
      manifest: |
        service_catalog:
          plans:
          - name: example-plan
            instance_groups:
            - name: example-instance-group
              vm_extensions: (( .broker.example_vm_extensions.value )) # add th
is line
```

3. Add the following to the `property_blueprints` section under the broker job:

   For example:

```
    property_blueprints:           # add this section
    - name: example_vm_extensions
      type: multi_select_options
      configurable: true
      optional: true
      options:
      - name: "public_ip"
        label: "Internet Connected VMs (on supported IaaS providers)"
```

(Optional) Enable Floating Stemcells

Ops Manager provides a feature called Floating Stemcells that allows PCF to quickly propagate a patched stemcell to all VMs in the deployment that have the same compatible stemcell. Both the broker deployment and the service instances deployed by the On-Demand Broker can make use of this feature. Enabling this feature can help ensure that all of your service instances are patched to the latest stemcell.

For the service instances to be installed with the latest stemcell automatically, ensure that the `upgrade-all-service-instances` errand is selected.

To enable floating stemcells for your tile, update your `tile.yml` file as follows:

1. Implement floating stemcells.

   For example:

```
job_types:
  templates:
  - name: broker
    manifest: |
      service_deployment:
        releases:
        - name: release-name
          version: 1.0.0
          jobs: [job_server]
```

```
        stemcells:
          - os: ubuntu-trusty
            version: (( $self.stemcell_version )) # Add this line
```

2. Configure the `stemcell_criteria`.

   For example:

```
---
name: example-on-demand-service
product_version: 1.0.0
stemcell_criteria:
  os: ubuntu-trusty
  version: '3312'
  enable_patch_security_updates: true  # Add this line
```

(Optional) Allow Secure Binding

You can give Ops Manager operators the option to enable secure binding. If secure binding is enabled, service instance credentials are stored securely in runtime CredHub. When users create bindings or service keys, ODB passes a secure reference to the service credentials through the network instead of plain text.

Note: To use the secure binding credentials feature you must use PCF v2.0 or later.

To include the option to enable secure binding, update your `tile.yml` file as follows:

1. Add `secure_binding_credentials` to the top-level properties block in the on-demand broker manifest.

   For example:

```
secure_binding_credentials:
  enabled: true
  authentication:
    uaa:
      client_id: CREDHUB_CLIENT_ID # client ID used by broker when communicatin
g with CredHub
      client_secret: CREDHUB_CLIENT_SECRET # client secret used by broker when
communicating with CredHub
      ca_cert: UAA_CA_CERT
```

2. To let users enable and disable this feature in the Ops Manager UI, you need to make some changes to your tile's metadata file:

   1. Add a form field to allow the user to enable/disable secure bindings. For an example form field, see the example-kafka-on-demand-tile.

   2. Add an element in `property_blueprints` that reads the setting in the form field and exposes the appropriate manifest snippet for CredHub and secure binding. For an example `property_blueprints` section, see the example-kafka-on-demand-tile.

3. Change the broker job so that it consumes the CredHub BOSH link from the `property_blueprints` section. For an example broker job, see the example-kafka-on-demand-tile.

4. Change the broker job so that it consumes the generated secure bindings manifest snippet. For an example broker job, see the example-kafka-on-demand-tile.

Create a pull request or raise an issue on the source for this page in GitHub

Service Adapter Interface Reference

This topic describes the subcommands used with the Service Adapter Interface.

Service Adapter Interface

Implement your service adapter as a binary. The service adapter receives its parameters as a JSON document using stdin.

For example service adapters, see the following examples written in golang:

- Redis
- Kafka

Note: The Redis and Kafka examples above use the SDK to help with cross-cutting concerns. For example, reading the JSON document from stdin.

A service adapter is expected to respond to the subcommands. For each of these subcommands, the following applies:

- An exit status of `0` indicates that the command succeeded.
- An exit status of `10` indicates not implemented.
- Any non-zero exit status indicates failure.

For a list of possible subcommands and the structure of the JSON document passed via stdin, see the subcommands below.

generate-manifest

This section contains the following topics:

- Input Parameters
- Output

Notes:

- The on-demand broker (ODB) requires `generate-manifest` to be **idempotent**. Given the same arguments when a previous manifest is supplied—which happens during a deployment update—the command should always output the same BOSH manifest.

- When determining whether there are pending changes for an instance during an update, ODB *ignores* any configuration supplied in the update block of the manifest returned by the `generate-manifest` subcommand.
- Service Authors should ensure that the service releases and stemcells satisfy the functional requirements of the service adapter. This can be achieved, for example, by checking that the service release satisfies a minimum version constraint.

Input Parameters

This section details the parameters provided to the `generate-manifest` subcommand using stdin.

See the following example:

```
{
  "generate_manifest": {
    "service_deployment": "SERVICE-DEPLOYMENT-JSON",
    "plan": "PLAN-JSON",
    "previous_plan": "PREVIOUS-PLAN-JSON",
    "previous_manifest": "PREVIOUS-MANIFEST-YAML",
    "request_parameters": "REQUEST-PARAMETERS-JSON",
    "previous_secrets": "PREVIOUS-SECRETS-JSON",
    "previous_configs": "PREVIOUS-CONFIGS-JSON"
  }
}
```

All arguments are passed as strings, not JSON objects.

For example:

```
{
  "generate_manifest": {
    "service_deployment": "{\"deployment_name\":\"some-name\"...}"
    // ...
  }
}
```

Note: Pivotal recommends that service authors use the following order of precedence in their service adapters when generating manifests:

1. Arbitrary parameters
2. Previous manifest properties
3. Plan properties

For an example, see `auto_create_topics` in the example Kafka service adapter.

SERVICE-DEPLOYMENT-JSON

`SERVICE-DEPLOYMENT-JSON` provides information regarding the BOSH Director.

The following table describes the JSON structure required for `SERVICE-DEPLOYMENT-JSON`:

Field	Type	Description
deployment_name	string	Name of the deployment on the Director, in the format `service-instance_GUID`
releases	array of releases	List of service releases configured for the deployment by the operator
release.name	string	Name of the release on the Director
release.version	string	Version of the release
release.jobs	array of strings	List of jobs required from the release
stemcells	array of stemcells	The stemcells available on the Director
stemcell.stemcell_os	string	Stemcell OS available on the Director
stemcell.stemcell_version	string	Stemcell version available on the Director

For example:

```
{
    "deployment_name": "service-instance_GUID",
    "releases": [{
        "name": "kafka",
        "version": "dev.42",
        "jobs": [
            "kafka_node",
            "zookeeper"
        ]
    }],
    "stemcells": [{
        "stemcell_os": "BeOS",
        "stemcell_version": "2"
    }, {
        "stemcell_os": "Windows",
        "stemcell_version": "3"
    }]
}
```

Keep in mind the following:

- ODB only supports using exact release and stemcell versions. The use of `latest` and floating stemcells are not supported.
- Your Service Adapter should be opinionated about which jobs it requires to generate its manifest. For example, the Kafka example requires `kafka_node` and `zookeeper`. It should not be opinionated about the mapping of BOSH release to job. The jobs can all be provided by one release or across many. The SDK provides the helper function `GenerateInstanceGroupsWithNoProperties` for generating instance groups without any properties. The Kafka example service adapter uses this helper function and invokes it to

map the service releases parameter to the BOSH manifest `releases` and `instance_groups` sections.

- You should provide documentation about which jobs are required by your Service Adapter, and which BOSH releases operators should get these jobs from.

PLAN-JSON

`PLAN-JSON` specifies the plan that the manifest is generated for.

The following table describes the schema of the JSON structure required for `PLAN-JSON`:

Field	Type	Description
instance_groups	array of instance groups	Instance groups configured for the plan
instance_group.name	string	Name of the instance group
instance_group.vm_type	string	The vm_type configured for the instance group, matches one in the cloud config on the director
instance_group.vm_extensions	array of strings	Optional, the vm_extensions configured for the instance group, must be present in the cloud config on the director
instance_group.persistent_disk_type	string	Optional, the persistent_disk_type configured for the instance group, matches one in the cloud config on the director
instance_group.networks	array of strings	The networks the instance group is supposed to be in
instance_group.instances	int	Number of instances for the instance group
instance_group.lifecycle	string	Optional, specifies the kind of workload the instance group represents. Valid values are `service` and `errand`; defaults to `service`
instance_group.azs	array of strings	A list of availability zones that the instance groups should be striped across
instance_group.migrated_from	array of migrations	Optional, list of bosh migrations
migration.name	string	Optional, name of the instance group to be migrated from
properties	map	Properties which the operator has configured for deployments of the current plan
lifecycle_errands	map	Optional, details of post-deploy and pre-delete errands
lifecycle_errands.post_deploy	array of errands	Optional, post-deploy errands configured for the plan
lifecycle_errands.pre_delete	array of errands	Optional, pre-delete errands configured for the plan
errand.name	string	Errand name

Field	Type	Description
errand.instances	array of strings	Optional, for a colocated errand, specify a list of INSTANCE-NAME/INSTANCE-IDX to run the errand
update	map	Update block which the operator has configured for deployments of the current plan
update.canaries	int	Plan-specific number of canary instances
update.max_in_flight	int	Plan-specific maximum number of non-canary instances to update in parallel
update.canary_watch_time	string	Plan-specific time in milliseconds that the BOSH Director sleeps before checking whether the canary instances are healthy
update.update_watch_time	string	Plan-specific time in milliseconds that the BOSH Director sleeps before checking whether the non-canary instances are healthy
update.serial	boolean	Optional, plan-specific flag to deploy instance groups sequentially (`true`), or in parallel (`false`); defaults to `true`

For example:

```
{
  "instance_groups": [
    {
      "name": "example-server",
      "vm_type": "small",
      "vm_extensions": [
        "some",
        "extensions"
      ],
      "persistent_disk_type": "ten",
      "networks": [
        "example-network"
      ],
      "azs": [
        "example-az"
      ],
      "instances": 1,
      "migrated_from": [
        {
          "name": "old-example-server"
        }
      ]
    },
    {
      "name": "example-migrations",
      "vm_type": "small",
      "persistent_disk_type": "ten",
      "networks": [
        "example-network"
      ],
      "instances": 1,
      "lifecycle": "errand"
    }
  ],
```

```
  "properties": {
    "example": "property"
  },
  "lifecycle_errands": {
    "post_deploy": [
      {
        "name": "health-check"
      },
      {
        "name": "init-replication",
        "instances": ["primary-node/0"]
      }
    ],
    "pre_delete": [
      {
        "name": "cleanup",
        "instances": ["example-server/0"]
      }
    ]
  },
  "update": {
    "canaries": 1,
    "max_in_flight": 2,
    "canary_watch_time": "1000-30000",
    "update_watch_time": "1000-30000",
    "serial": true
  }
}
```

Plans are composed by the operator and consist of resource mappings, properties, and an optional update block.

Resource Mappings

The `instance_groups` section of the plan JSON. This maps service deployment instance groups (defined by the service author) to resources (defined by the operator).

You should document the list of instance group names required for a deployment, for example, “redis-server”. You should also document any recommended resource constraints. For example, operators must add a persistent disk if the persistence property is enabled. You can enforce these constraints in code.

The `instance_groups` section also contains a field for `lifecycle`, which can be set by the operator. The service adapter adds a `lifecycle` field to the instance group within the BOSH manifest when specified.

Properties

Properties are service-specific parameters that you choose. The Redis example exposes a property `persistence`, which takes a boolean value and toggles disk persistence for Redis. You should document these properties for the operator.

(Optional) Update Block

This block defines a plan-specific configuration for BOSH' s update instance operation. Although the ODB considers this block optional, the service adapter must output an update block in every manifest it generates. Some ways to achieve that are:

1. *(Recommended)* Define a default update block for all plans, which is used when a plan-specific update block is not provided by the operator.
2. Hard code an update block for all plans in the service adapter.
3. Make the update block mandatory, so that operators must provide an update block for every plan in the service catalog section of the ODB manifest.

REQUEST-PARAMS-JSON

This is a JSON object that holds the entire body of the service provision or service update request sent by the Cloud Controller to the service broker. The request parameters JSON will be `null` for upgrades.

The field `context` holds platform-specific contextual information under which the service instance is to be provisioned.

The field `parameters` contains arbitrary key-value pairs that were passed by the application developer as a cf CLI parameter when creating or updating the service instance. They allow Cloud Foundry users to override the default configuration for a service plan. For example, the Kafka service adapter supports the `auto_create_topics` arbitrary parameter to configure auto-creation of topics on the cluster.

Note: When updating an existing service instance, any arbitrary parameters passed on a previous create or update are not passed again. Therefore, for arbitrary parameters to stay the same across multiple deployments they must be retrieved from the previous manifest.

For example:

```
{
  "context": {
    "platform": "cloudfoundry",
    "some_field": "some-contextual-data"
  },
  "organization_guid": "org-guid-here",
  "parameters": {
    "parameter1": {
      "sub-param1": 1,
      "sub-param2": "some-info"
    }
  },
  "plan_id": "plan-id-here",
  "service_id": "service-id-here",
  "space_guid": "space-guid-here"
}
```

PREVIOUS-MANIFEST-YAML

`PREVIOUS-MANIFEST-YAML` represents the previous BOSH deployment manifest for the service instance. If you have a new deployment, the YAML file is empty.

The manifest format matches the BOSH v2 manifest. For more information about the BOSH v2 manifest, see Deployment Config in the BOSH documentation.

The service author must perform any necessary service-specific migration logic if previous manifest is non-nil.

Another use case of the previous manifest is for the migration of deployment properties which need to stay the same across multiple deployments of a manifest. In the Redis example, we generate a password when we do a new deployment. But, when the previous deployment manifest is provided, we copy the password over from the previous deployment, because generating a new password for existing deployments will break existing bindings.

For an example, see the example Redis service adapter.

WARNING: If the service adapter does not migrate properties from the old manifest to the new one, the update fails.

PREVIOUS-PLAN-JSON

This argument takes the previous plan as a JSON string.

The previous plan is nil if this is a new deployment.

The format of the plan should match the plan schema. The previous plan can be used for complex plan migration logic. For an example, the Kafka service adapter rejects a plan migration if the new plan reduces the number of instances, to prevent data loss.

PREVIOUS-SECRETS-JSON

If `enable_secure_manifests` is set to `true` in the broker, any secrets that use references to BOSH-generated variables or literal CredHub paths in the previous service instance manifest are resolved and sent to the adapter in the `PREVIOUS-SECRETS-JSON` parameter.

These secrets are passed during updates but not during upgrades.

If `enable_secure_manifests` is set to `false` in the broker, then the `PREVIOUS-SECRETS-JSON` parameter is empty.

The following is an example previous service instance manifest:

```
...
password: ((redis_password))
root_ca: ((/global/root_ca))

variables:
- name: redis_password
  type: password
```

The service instance manifest snippet above produces a secrets JSON parameter similar to the

following. The keys are the reference names, and the values are the resolved secrets:

```
{
  "((redis_password))": "some-bosh-generated-password",
  "((/global/root_ca))": "some-global-value"
}
```

Note: You can find the secrets key by accessing the manifest field that contains the reference to the variable.

PREVIOUS-CONFIGS-JSON

This argument provides the previous BOSH configs specified for the service instance. If populated, it will contain a map of config types to config content. For example:

```
{
  "cloud":"
    vm_types:
     - name: my-service-instance-small
       cloud_properties:
         cpu: 1"
}
```

Output

The following table describes the supported exit codes and output for the `generate-manifest` subcommand:

Exit code	Description	Output
0	success	• Stdout: JSON document containing the BOSH manifest YAML, a map of adapter-generated secrets to be managed by ODB, and a map of BOSH configs
10	not implemented	
anything else	failure	• Stdout: optional error message for CF CLI users • Stderr: error message for operator • ODB logs both stdout and stderr

Example JSON output printed when the `generate-manifest` command is successful:

```
{
  "manifest": "GENERATED-BOSH-MANIFEST-YAML",
  "secrets": { "secret1":"value1", "secret2":"value2" },
  "configs": {
    "cloud":"
       vm_types:
       - name: my-service-instance-small
```

```
          cloud_properties:
            cpu: 1"
  }
}
```

dashboard-url

This section contains the following topics:

- Input Parameters
- Output

Input Parameters

The following section details the parameters provided to the `dashboard-url` subcommand using stdin.

See the following example:

```
{
  "dashboard_url": {
    "instance_id": "SERVICE-INSTANCE-ID",
    "plan": "PLAN-JSON",
    "manifest": "MANIFEST-YAML"
  }
}
```

All the arguments are passed as strings and not JSON objects.

For example:

```
{
  "dashboard_url": {
    ...
    "manifest": "---\nname: my-service-instance\n..."
  }
}
```

SERVICE-INSTANCE-ID

This parameter is the unique identifier of the service instance provided by the Cloud Controller. For example, `42a09f38-c15b-47fe-a24e-ebf5f83ebd0`.

PLAN-JSON

This parameter is the current plan for the service instance as JSON. The structure should be the same as the plan given in the generate manifest.

See the following example:

```
{
  "properties": {
    "persistence": true
```

```
  },
  "lifecycle_errands": {
    "post_deploy": [],
    "pre_delete": []
  },
  "instance_groups": [
    {
      "name": "my-example-server",
      "vm_type": "t2.small",
      "persistent_disk_type": "10GB",
      "instances": 1,
      "networks": [
        "default"
      ],
      "azs": [
        "z1"
      ]
    }
  ]
}
```

MANIFEST-YAML

This parameter is the current manifest as YAML.

The manifest format matches the BOSH v2 manifest. For more information about the BOSH v2 manifest, see Deployment Config in the BOSH documentation.

See the following example:

```
name: my-service-instance
releases:
- name: my-service
  version: 1.1.0
stemcells:
- alias: only-stemcell
  os: ubuntu-trusty
  version: "3468.1"
instance_groups:
- name: my-example-server
  instances: 1
  jobs:
  - name:  my-example-server
    release: my-service
  vm_type: t2.small
  stemcell: only-stemcell
  persistent_disk_type: 10GB
  azs:
  - z1
  networks:
  - name: default
  properties:
    some-parameter:
      param1: "some-value"
      param2: 1
update:
  canaries: 4
  canary_watch_time: 30000-240000
```

```
  update_watch_time: 30000-240000
  max_in_flight: 4
tags:
  product: my-product
addons:
- name: some-addon
  jobs:
  - name: my-example-server
    release: my-service
```

Output

The following table describes the supported exit codes and output for the `dashboard-url` subcommand:

Exit code	Description	Output
0	success	• Stdout: dashboard URL JSON
10	not implemented	
anything else	failure	• Stdout: optional error message for CF CLI users • Stderr: error message for operator ODB logs both stdout and stderr

Example JSON output printed when the `dashboard-url` command is successful:

```
{
   "dashboard_url":"https://someurl.example.com"
}
```

The following table describes the output JSON above:

Field	Type	Description
dashboard_url	string	Dashboard url returned to the cf user

create-binding

This section contains the following topics:

- Input Parameters
- Output

Input Parameters

The following section details the parameters required by the `create-binding` subcommand as a JSON document using stdin.

See the following example:

```
{
  "create_binding": {
    "binding_id": "BINDING-ID",
    "bosh_vms": "BOSH-VMS-JSON",
    "manifest": "MANIFEST-YAML",
    "request_parameters": "REQUEST-PARAMETERS-JSON",
    "secrets": "MANIFEST-SECRETS-JSON",
    "dns_addresses": "DNS-ADDRESSES-JSON"
  }
}
```

All the arguments are passed as strings and not JSON objects.

For example:

```
{
  "create_binding": {
    // ...
    "bosh_vms": "{\"mysql_node\": [\"192.0.2.1\", \"192.0.2.2\", \"192.0.2.3\"]}"
  }
}
```

BINDING-ID

This parameter is the binding ID generated by the Cloud Controller.

BOSH-VMS-JSON

This parameter is a JSON map of instance group name to an array of IP addresses provisioned for that instance group.

See the following example:

```
{
  "mysql_node": ["192.0.2.1", "192.0.2.2", "192.0.2.3"],
  "management_box": ["192.0.2.4"]
}
```

This can be used to connect to the instance deployment, if required, or to create a service specific binding. In the example above, the Service Adapter may connect to MySQL as the admin and create a user. As part of the binding, the `mysql_node` IP addresses would be returned, but not the `management_box`.

MANIFEST-YAML

This parameter is the current manifest as YAML. This is used to extract information about the deployment that is necessary for the binding, such as admin credentials with which to create users.

The manifest format matches the BOSH v2 manifest. For more information about the BOSH v2 manifest, see Deployment Config in the BOSH documentation.

REQUEST-PARAMS-JSON

This parameter is a JSON object that holds the entire body of the service binding request sent by the Cloud Controller to the service broker.

The field `bind_resource` contains key-value pairs for `app_guid`, `credential_client_id` and `route`. If using the golang SDK, the `brokerapi.BindResource` struct containing these fields can be accessed using the `BindResource()` helper method on requestParams.

The field `parameters` contains arbitrary key-value pairs which were passed by the app developer as a `cf` CLI parameter when creating, or updating the service instance. If using the golang SDK, it can be obtained using the `ArbitraryParams()` helper method on requestParams.

See the following example:

```
{
  "app_guid": "app-guid-here",
  "bind_resource": {
    "app_guid": "app-guid-here"
  },
  "context": {
    "platform": "cloudfoundry",
    "some_param": "some-value"
  },
  "parameters": {
    "parameter1": {
      "sub-param1": 1,
      "sub-param2": "some-info"
    }
  },
  "plan_id": "my-plan",
  "service_id": "my-service"
  }
```

MANIFEST-SECRETS-JSON

If `enable_secure_manifests` is set to `true` in the broker, any secrets in the service instance manifest that use references to BOSH-generated variables or literal CredHub paths are resolved and sent to the adapter in the `MANIFEST-SECRETS-JSON` parameter.

If `enable_secure_manifests` is set to `false` in the broker, then the `MANIFEST-SECRETS-JSON` parameter is empty.

The following is an example service instance manifest:

```
...
password: ((redis_password))
root_ca: ((/global/root_ca))

variables:
- name: redis_password
  type: password
```

The service instance manifest snippet above produces a secrets JSON parameter similar to the

following. The keys are the reference names, and the values are the resolved secrets:

```
{
  "((redis_password))": "some-bosh-generated-password",
  "((/global/root_ca))": "some-global-value"
}
```

Note: You can find the secrets key by accessing the manifest field that contains the reference to the variable.

DNS-ADDRESSES-JSON

When this feature is enabled and a broker is deployed with `binding_with_dns` set for a plan, the ODB retrieves DNS addresses from BOSH before calling the adapter. ODB passes these addresses to the service adapter along with the names given in the `binding_with_dns` properties.

For how to enable this feature for your on-demand service, see Enable ODB to Obtain BOSH DNS Addresses.

The following is an example `binding_with_dns` configuration in the broker manifest:

```
plans:
  ...
  - name: example-plan
    binding_with_dns:
    - name: leader-address
      link_provider: example-link-1
      instance_group: leader-node
    - name: follower-address
      link_provider: example-link-2
      instance_group: follower-node
```

The snippet above produces the following in `DNS-ADDRESSES-JSON`:

```
{
  "leader-address": "q-s0.leader-node.default.service-instance_c1371314-643f-48b7-b80a
-6741e7377022.bosh",
  "follower-address": "q-s0.follower-node.default.service-instance_c1371314-643f-48b7-
b80a-6741e7377022.bosh"
}
```

Each entry in `binding_with_dns` is converted to a single BOSH DNS address using the BOSH links API.

The On Demand Services SDK includes the `DNSAddresses` parameter for the `CreateBinding` and `DeleteBinding` methods. The ODB invokes the `CreateBinding` and `DeleteBinding` methods with this parameter, which is a map of name to DNS address.

Output

The following table describes the supported exit codes and output for the `create-binding`

subcommand:

Exit code	Description	Output
0	success	• Stdout: binding credentials JSON
10	subcommand not implemented	
42	app_guid not provided in the binding request body	• Stderr: error message for operator • ODB logs both stdout and stderr
49	binding already exists	• Stderr: error message for operator • ODB logs both stdout and stderr
anything else	failure	• Stdout: optional error message for CF CLI users • Stderr: error message for operator • ODB logs both stdout and stderr

Example JSON output printed when the `create-binding` command is successful:

```
{
  "credentials": {
    "username": "user1",
    "password": "reallysecret"
  },
  "syslog_drain_url": "optional: for syslog drain services only",
  "route_service_url": "optional: for route services only"
}
```

delete-binding

This subcommand should invalidate the credentials that were generated by `create-binding` if possible. For example, the subcommand would delete the binding user in MySQL.

This section contains the following topics:

- Input Parameters
- Output

Input Parameters

This section describes the parameters required by the `delete-binding` subcommand.

See the following example:

```
{
  "delete_binding": {
    "binding_id": "BINDING-ID",
    "bosh_vms": "BOSH-VMS-JSON",
    "manifest": "MANIFEST-YAML",
```

```
    "delete_parameters": "DELETE-PARAMETERS-JSON",
    "secrets": "MANIFEST-SECRETS-JSON",
    "dns_addresses": "DNS-ADDRESSES-JSON"
  }
}
```

All the arguments are passed as strings and not JSON objects.

For example:

```
{
  "delete_binding": {
    // ...
    "manifest": "---\nname: some-name\n..."
  }
}
```

BINDING-ID

This parameter is the binding to be deleted.

BOSH-VMS-JSON

This parameter is a map of instance group name to an array of IPs provisioned for that instance group.

See the following example:

```
{
  "my-instance-group": ["192.0.2.1", "192.0.2.2", "192.0.2.3"]
}
```

MANIFEST-YAML

`MANIFEST-YAML` represents the parameter for the current manifest. BOSH uses the manifest to extract information about the deployment such as the credentials.

The manifest format matches the BOSH v2 manifest. For more information about the BOSH v2 manifest, see Deployment Config in the BOSH documentation.

For an example, see the Kafka delete binding.

DELETE-PARAMS-JSON

This parameter is a JSON object that holds query string parameters as useful hints for service brokers. For more information, see the Open Service Broker API documentation.

See the following example:

```
{
  "plan_id":"my-plan-id",
  "service_id":"my-service-id"
```

```
}
```

Note: This parameter is different from the create-binding `request_parameters` parameter and, in particular, does not include `parameters` or `bind_resource`.

MANIFEST-SECRETS-JSON

See MANIFEST-SECRETS-JSON above.

DNS-ADDRESSES-JSON

See DNS-ADDRESSES-JSON above.

Output

The following table describes the supported exit codes and output for the `delete-binding` subcommand:

exit code	Description	Output
0	success	• No output is required
10	not implemented	
41	binding does not exist	• Stderr: error message for operator • ODB logs both stdout and stderr
anything else	failure	• Stdout: optional error message for CF CLI users • Stderr: error message for operator • ODB logs both stdout and stderr

This can be used to connect to the actual VMs if required, to delete a service specific binding. For example, this can be used to delete a user in MySQL.

generate-plan-schemas

The broker uses the schema returned by this subcommand to validate service-specific configuration parameters. Apps Manager uses the schema to generate a form that users can use to populate those parameters. The schema must be in the JSON Schema draft-04 format. For more information about the plan schema, see the Open Service Broker API (OSBAPI) v2.13 specification.

If you do not want to validate all parameters and want additional parameters to be accepted without constraints, set the JSON schema field `additionalProperties` to `true`. For the location of this field, see the Kafka example adapter. For example, you might use this if you want to accept an undocumented optional parameter, for administration purposes, that should not be exposed through the Apps Manager UI.

This section contains the following topics:

Input Parameters

This section describes the parameters required by the `generate-plan-schemas` subcommand, passed using stdin.

See the following example:

```
{
  "generate_plan_schemas": {
    "plan": "PLAN-JSON"
  }
}
```

All the arguments are passed as strings and not JSON objects.

For example:

```
{
  "generate_plan_schemas": {
    "plan": "{\"instance_groups\":[]}"
  }
}
```

PLAN-JSON

This parameter is the service plan as JSON required to generate the JSON schema.

See the following example:

```
{
  "properties": {
    "persistence": true
  },
  "lifecycle_errands": {
    "post_deploy": [],
    "pre_delete": []
  },
  "instance_groups": [
    {
      "name": "my-example-server",
      "vm_type": "t2.small",
      "persistent_disk_type": "10GB",
      "instances": 1,
      "networks": [
        "default"
      ],
      "azs": [
        "z1"
      ]
    }
  ]
```

```
}
```

Output

The following table describes the supported exit codes and output for the `generate-plan-schemas` subcommand:

exit code	Description	Output
0	success	• Stdout: JSON document containing the plan schemas
10	not implemented	
anything else	failure	• Stdout: optional error message for cf CLI users • Stderr: error message for operator • ODB logs both stdout and stderr

Example JSON output printed when the `generate-plan-schemas` command is successful:

```
{
  "service_instance": {
    "create": {
      "parameters": {
        "": "http://json-schema.org/draft-04/schema#",
        "additionalProperties": true,
        "properties": {
          "auto_create_topics": {
            "description": "Auto create topics",
            "type": "boolean"
          },
          "default_replication_factor": {
            "description": "Replication factor",
            "type": "integer"
          }
        },
        "type": "object"
      }
    },
    "update": {
      "parameters": {
        "": "http://json-schema.org/draft-04/schema#",
        "additionalProperties": true,
        "properties": {
          "auto_create_topics": {
            "description": "Auto create topics",
            "type": "boolean"
          },
          "default_replication_factor": {
            "description": "Replication factor",
            "type": "integer"
          }
        },
        "type": "object"
      }
    }
```

```
  },
  "service_binding": {
    "create": {
      "parameters": {
        "": "http://json-schema.org/draft-04/schema#",
        "additionalProperties": false,
        "properties": {
          "topic": {
            "description": "The name of the topic",
            "type": "string"
          }
        },
        "type": "object"
      }
    }
  }
}
```

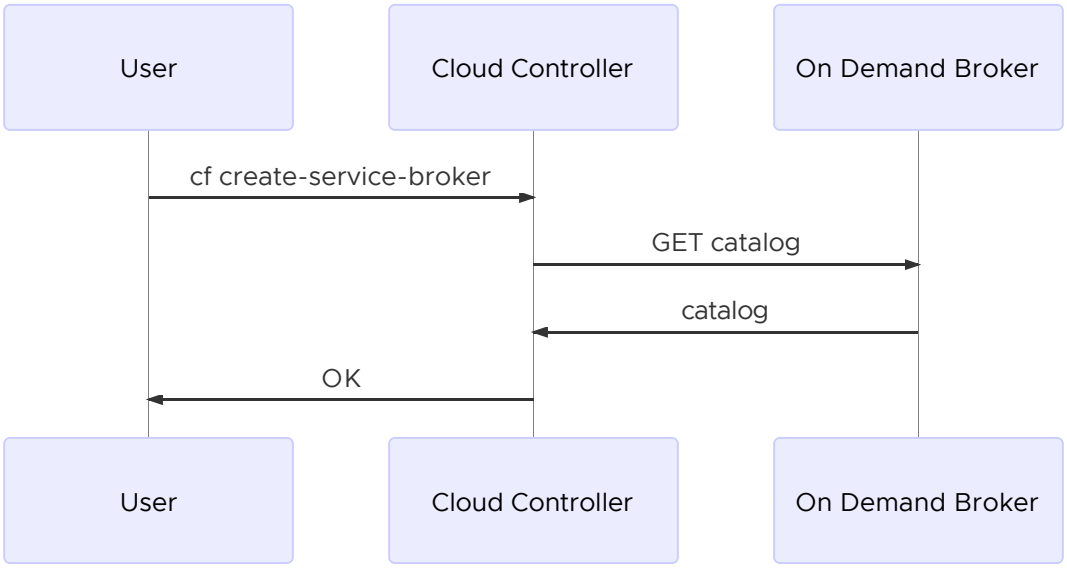
Create a pull request or raise an issue on the source for this page in GitHub

How On-Demand Services Process Commands

The sequence diagrams in this topic show how an on-demand service sets up and maintains service instances. The diagrams indicate which tasks are undertaken by the on-demand broker (ODB) and which require interaction with the Service Adapter.

Register the Service Broker with Cloud Foundry

The sequence diagram below shows the workflow for registering a service broker with Cloud Foundry.



About Creating and Updating Service Instances

This section contains diagrams that present the workflow for the following actions:

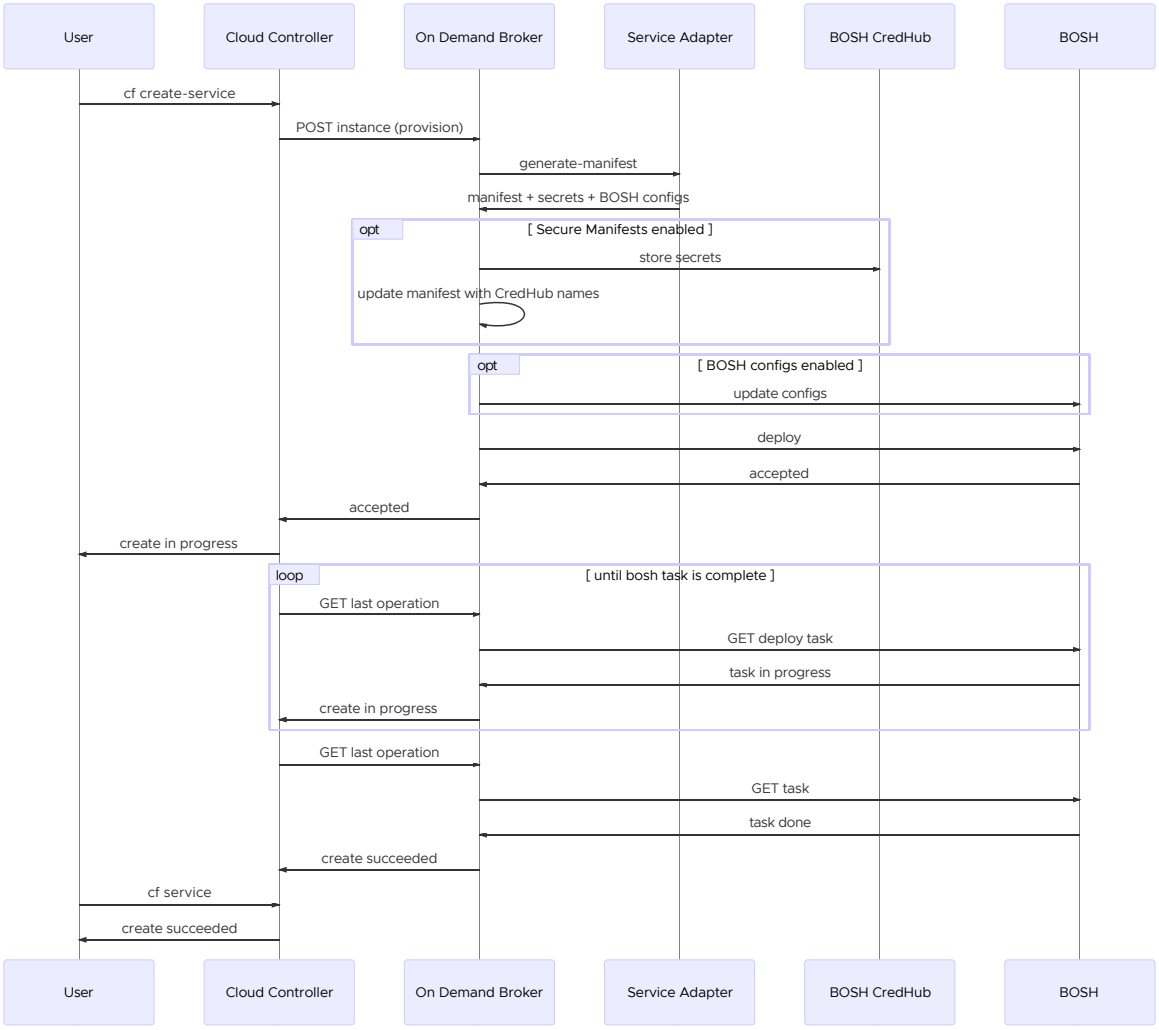
- Create a Service Instance

- Update a Service Instance
- Create or Update a Service Instance with Post-Deploy Errands
- Recreate All Service Instances

Create a Service Instance

To create a service instance, users run the `cf create-service` command. For more information about this command, see Creating Service Instances.

The sequence diagram below shows the workflow for creating a service instance.



There are two ways this process can fail:

- **Synchronously:** The Cloud Controller deletes the service according to its orphan mitigation strategy. For more information, see Orphans.
- **Asynchronously:** This happens while BOSH deploys the service instance. The Cloud Controller does not issue a delete request.

Update a Service Instance

To update a service instance, users run the `cf update-service` command. For more information about this command, see Update a Service Instance.

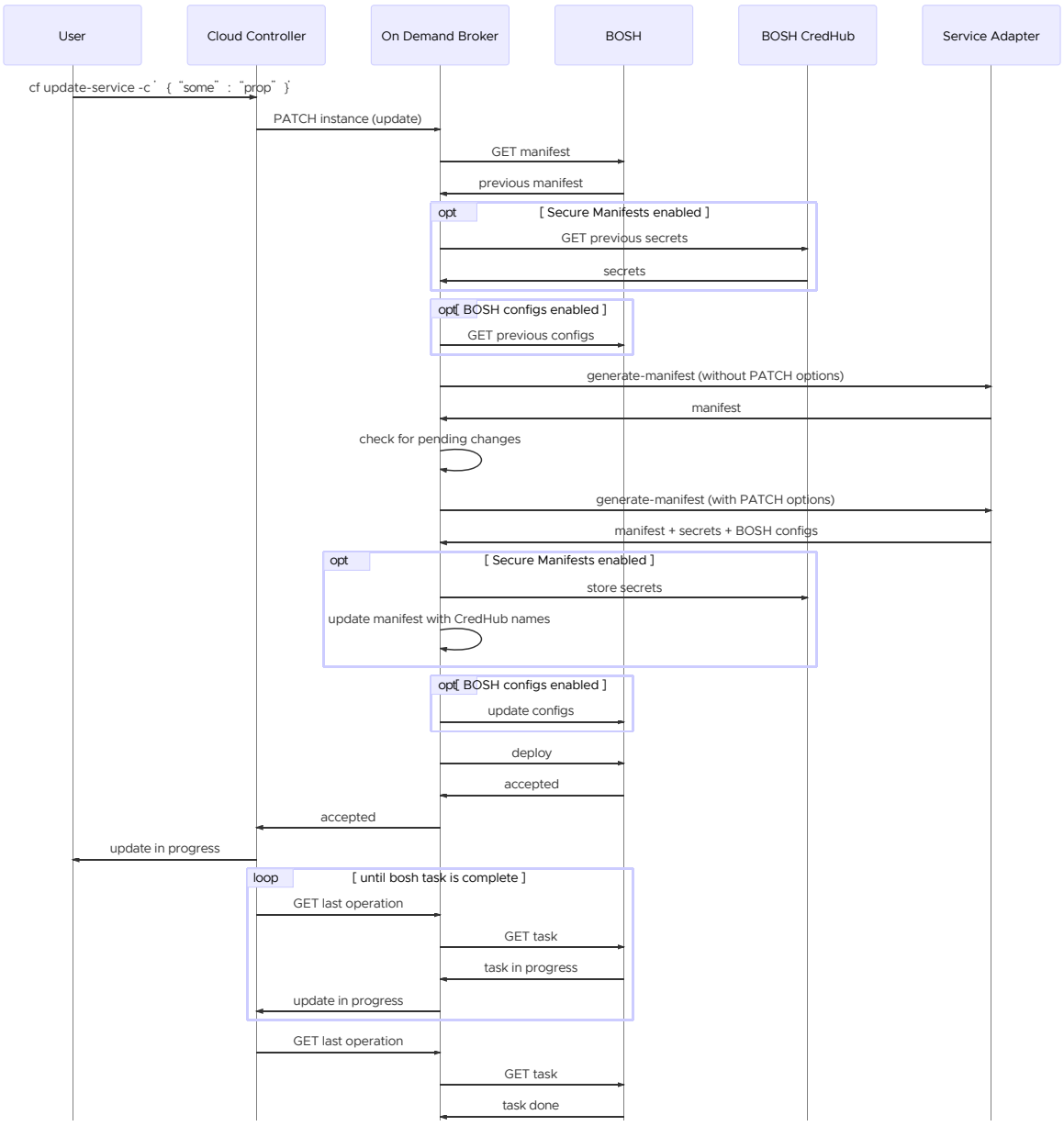
Updates can only proceed if the existing service instance is up-to-date. ODB calls `generate-manifest` on the service adapter to determine whether there are any pending changes for the instance.

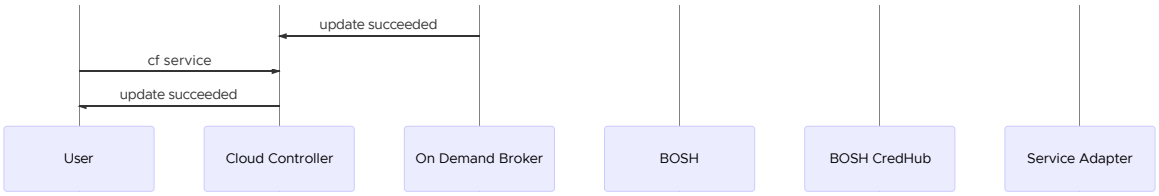
Note: When determining whether there are pending changes for an instance during an update, ODB ignores any configuration supplied in the update block of the manifest returned by the service adapter's `generate-manifest` subcommand For more information, see Update Block in the Cloud Foundry BOSH documentation.

Update When There Are No Pending Changes

If there are no pending changes, the update proceeds. The manifest from the second call to `generate-manifest` is deployed.

The sequence diagram below shows the workflow for updating a service instance if there are no pending changes.

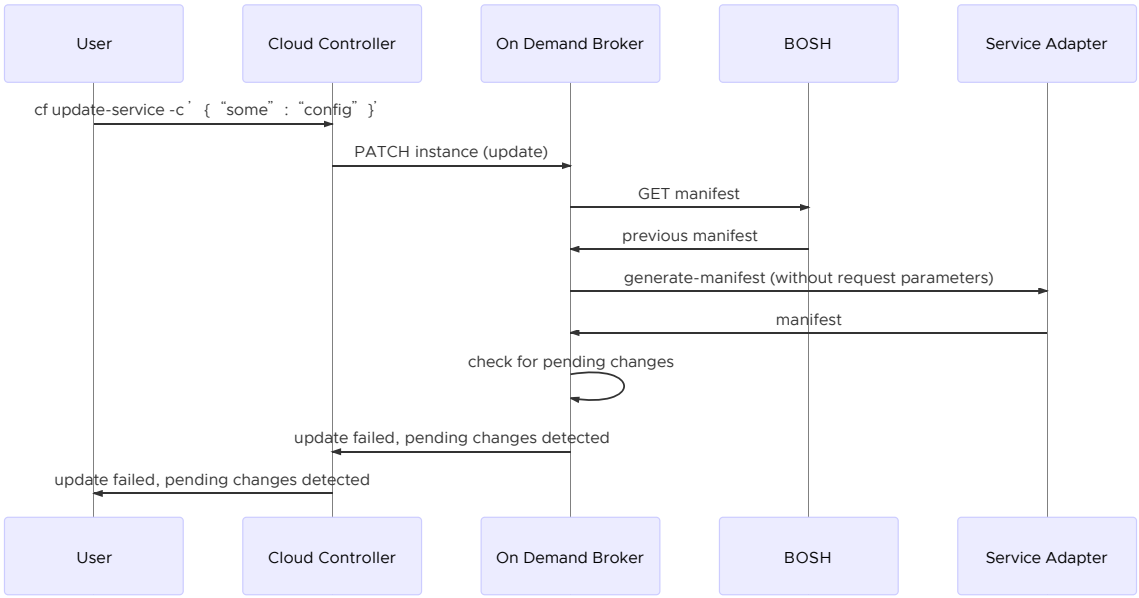




Update When There Are Pending Changes

If there are pending changes, the update fails.

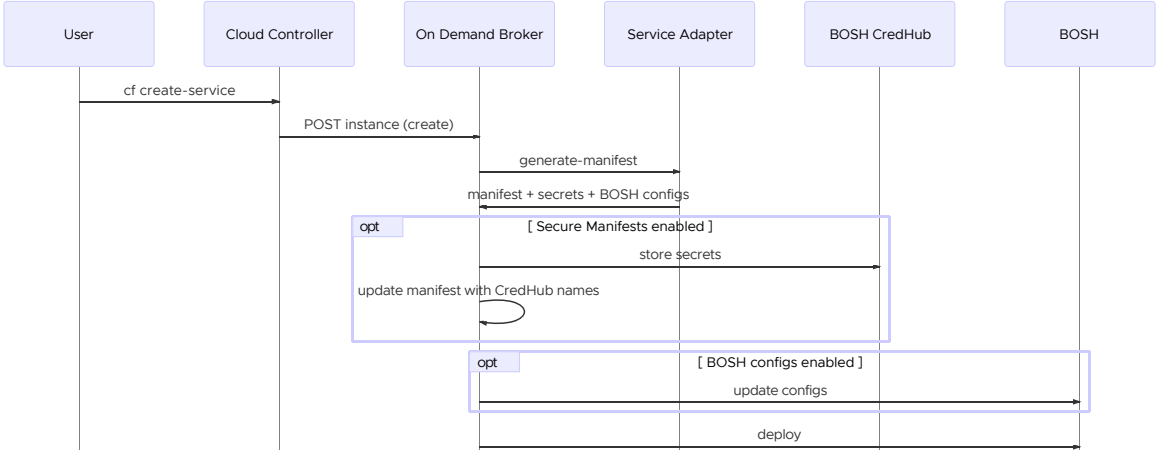
The sequence diagram below shows the workflow for updating a service instance if there are pending changes.

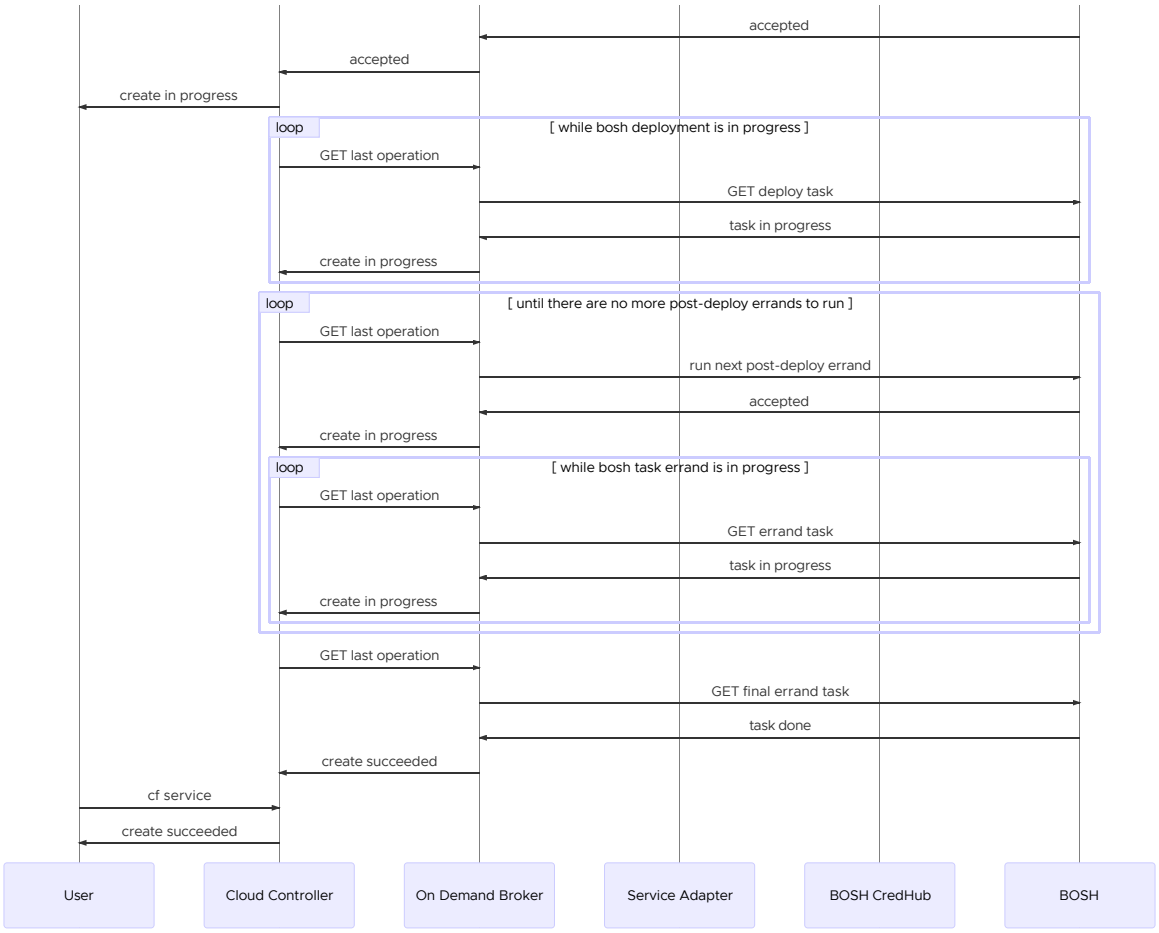


Create or Update a Service Instance with Post-Deploy Errands

If a user runs the `cf create-service` command with post-deploy errands configured for the deployment, ODB does not report success to Cloud Foundry until the deployment is created, or updated, and all post-deploy errands complete. For more information about post-deploy errands, see Service Instance Lifecycle Errands.

The sequence diagram below shows the workflow for creating or updating a service instance when post-deploy errands are configured.

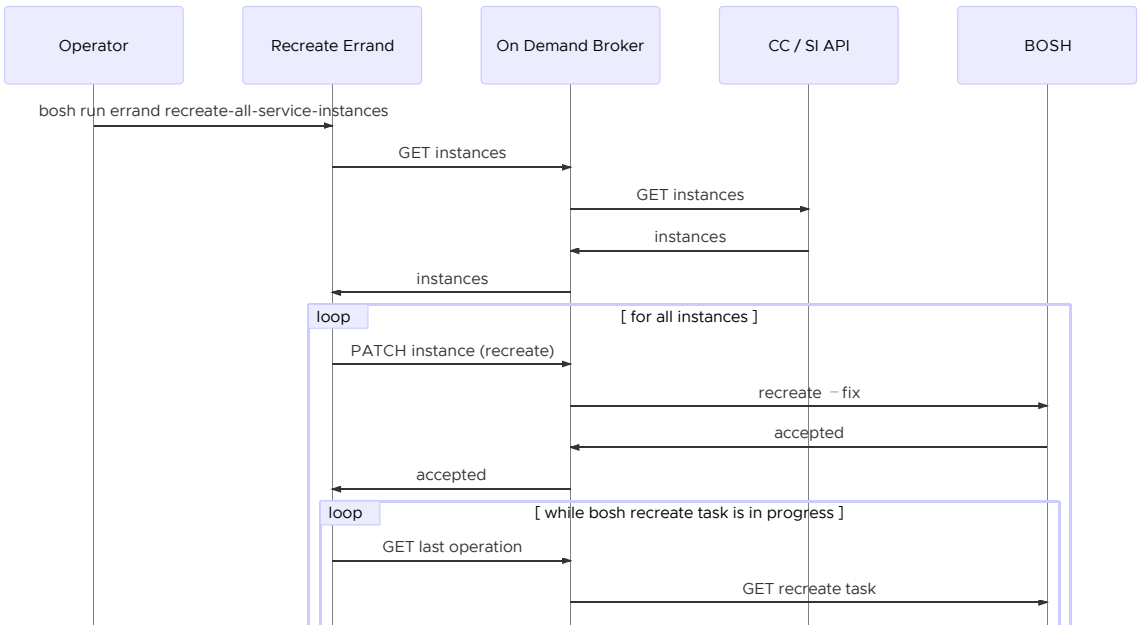


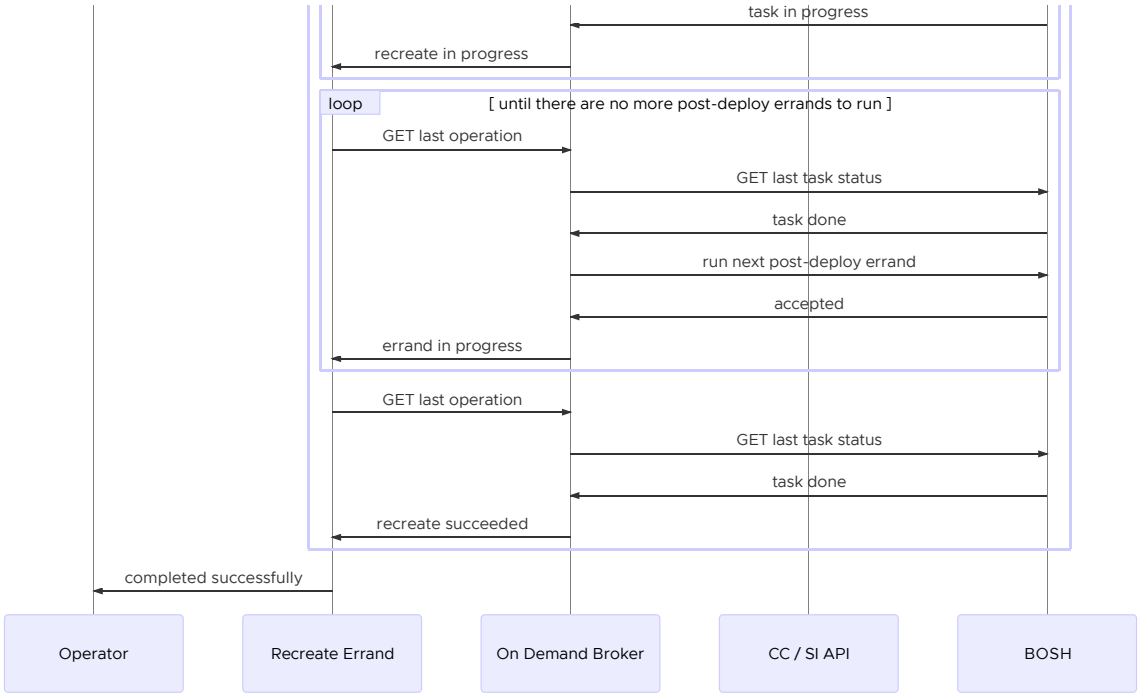


Recreate All Service Instances

ODB provides the BOSH errand `recreate-all-service-instances`. This errand executes a `bosh -d DEPLOYMENT-NAME recreate --fix` on each service instance (SI) managed by the broker. It is used for triggering low-level BOSH agent certificate re-installation, or for backup and restore purposes, for example in a migration between foundations.

The sequence diagram below shows the workflow for recreating service instances.





About Upgrading Service Instances

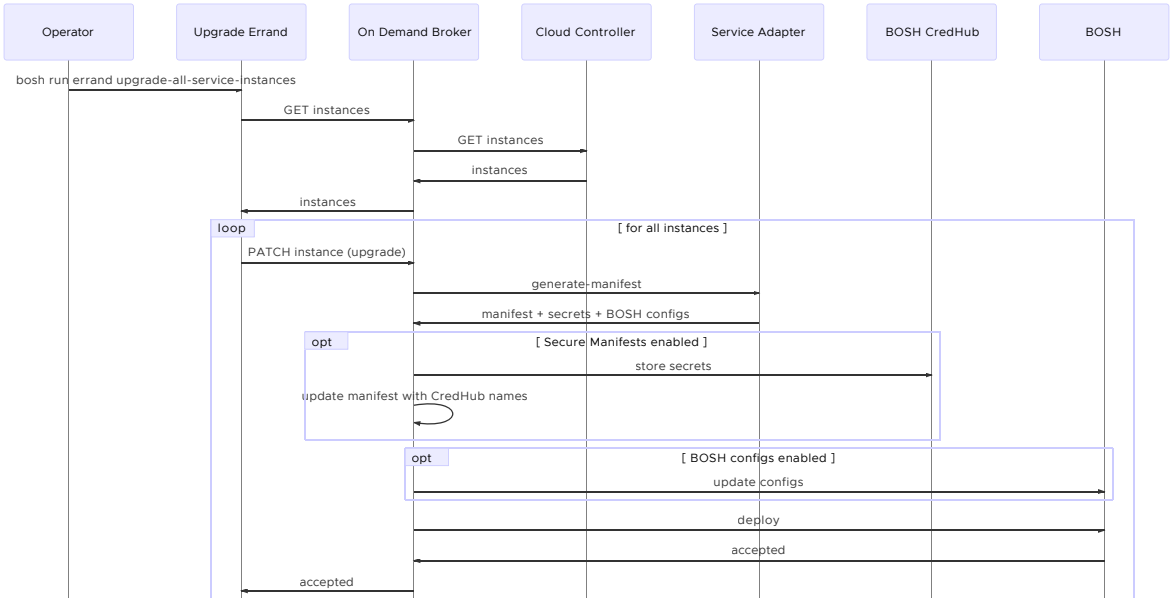
This section contains diagrams that present the workflow for the following actions:

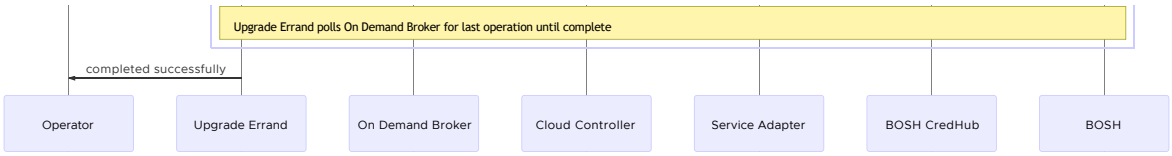
- Upgrade All Service Instance
- Upgrade All Service Instances with External Service Instances API Configured

Upgrade All Service Instances

ODB provides the BOSH errand `upgrade-all-service-instances`. This errand upgrades all service instances managed by the broker. This is also used when a plan changes. The errand updates all instances that implement a plan with the new plan definition. For more information, see Upgrade All Service Instances in the Operator Guide.

The sequence diagram below shows the workflow for upgrading all service instances.

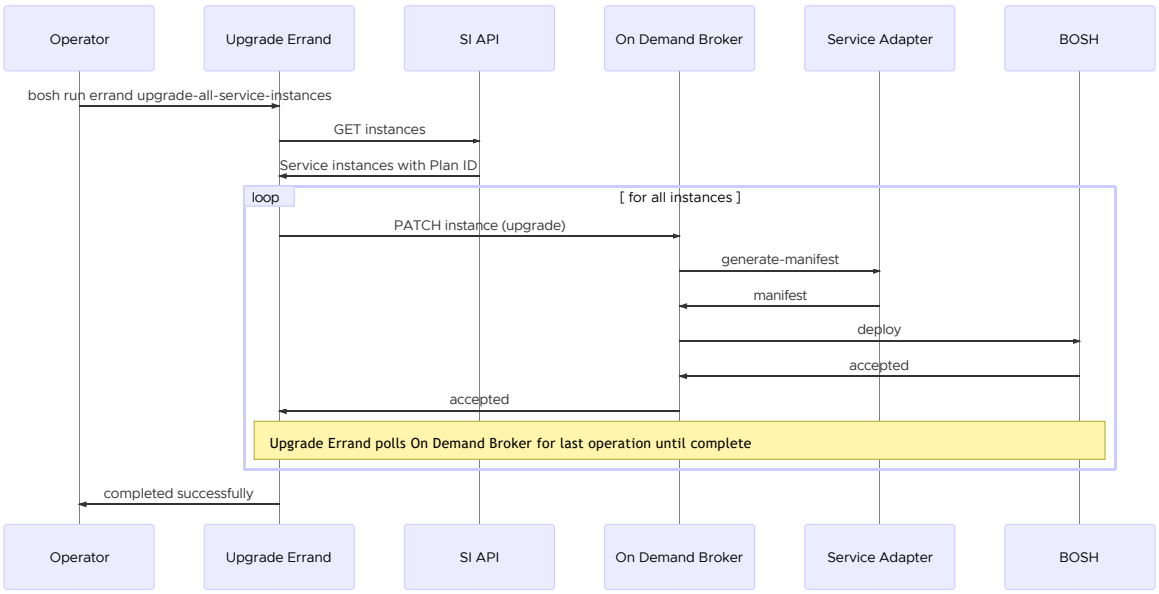




Upgrade All Service Instances with External Service Instances API Configured

If the service instances API is configured, the `upgrade-all-service-instances` errand connects to a different endpoint to gather the list of instances to upgrade. For more information, see Service Instances API.

The sequence diagram below shows the workflow for upgrading all service instances with external service instances API configured.



About Binding and Unbinding Service Instances

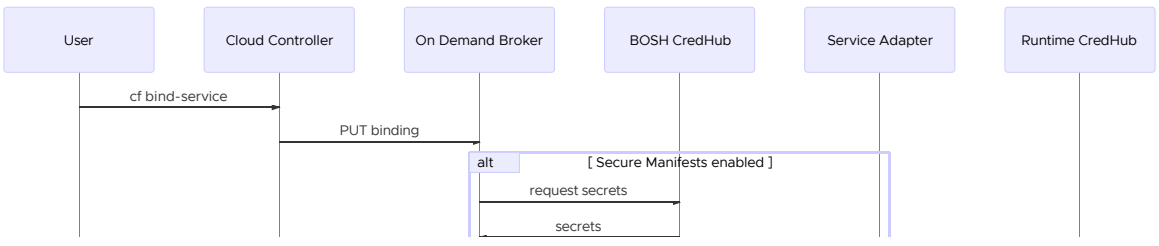
This section contains diagrams that present the workflow for the following actions:

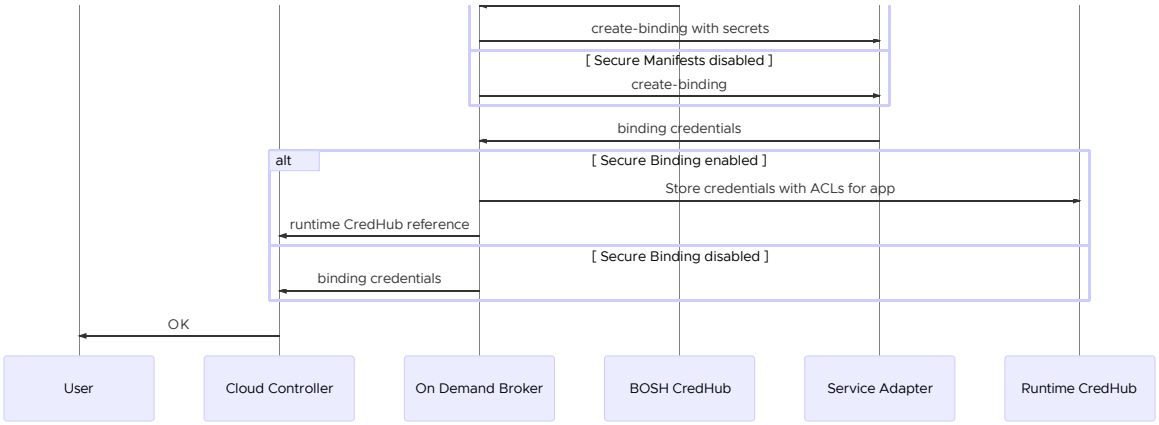
- Bind a Service Instance
- Unbind a Service Instance

Bind a Service Instance

To bind a service instance, users run the `cf bind-service` command. For more information about this command, see Bind a Service Instance.

The sequence diagram below shows the workflow for creating a binding.

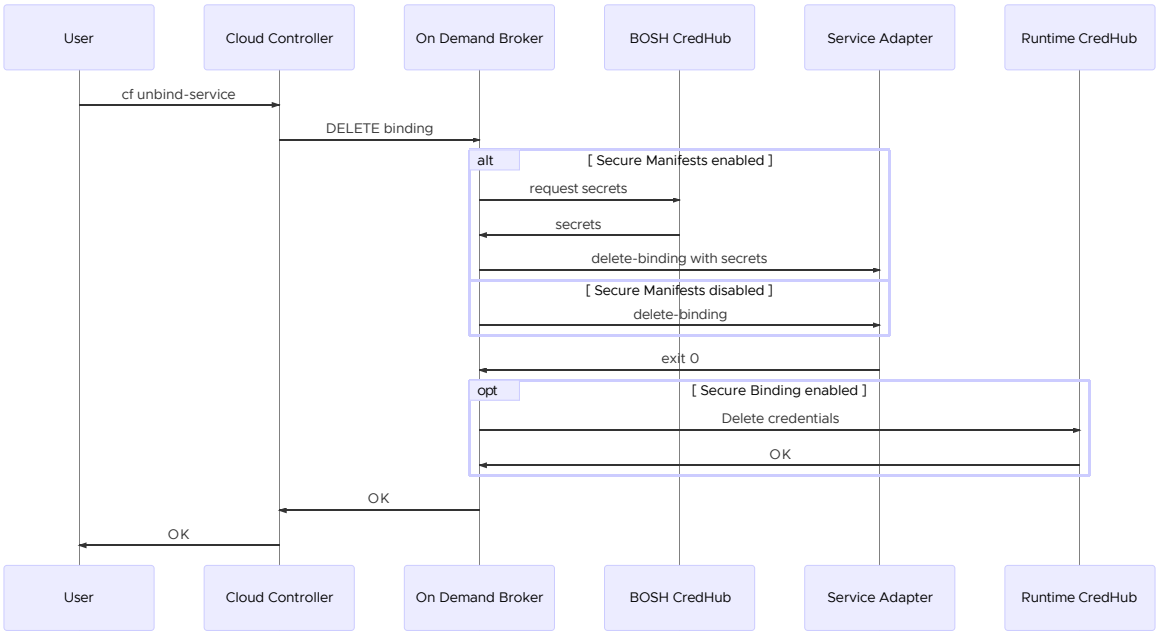




Unbind a Service Instance

To unbind a service instance, users run the `cf unbind-service` command. For more information about this command, see Unbind a Service Instance.

The sequence diagram below shows the workflow for unbinding a service instance.



About Deleting Service Instances

This section contains diagrams that present the workflow for the following actions:

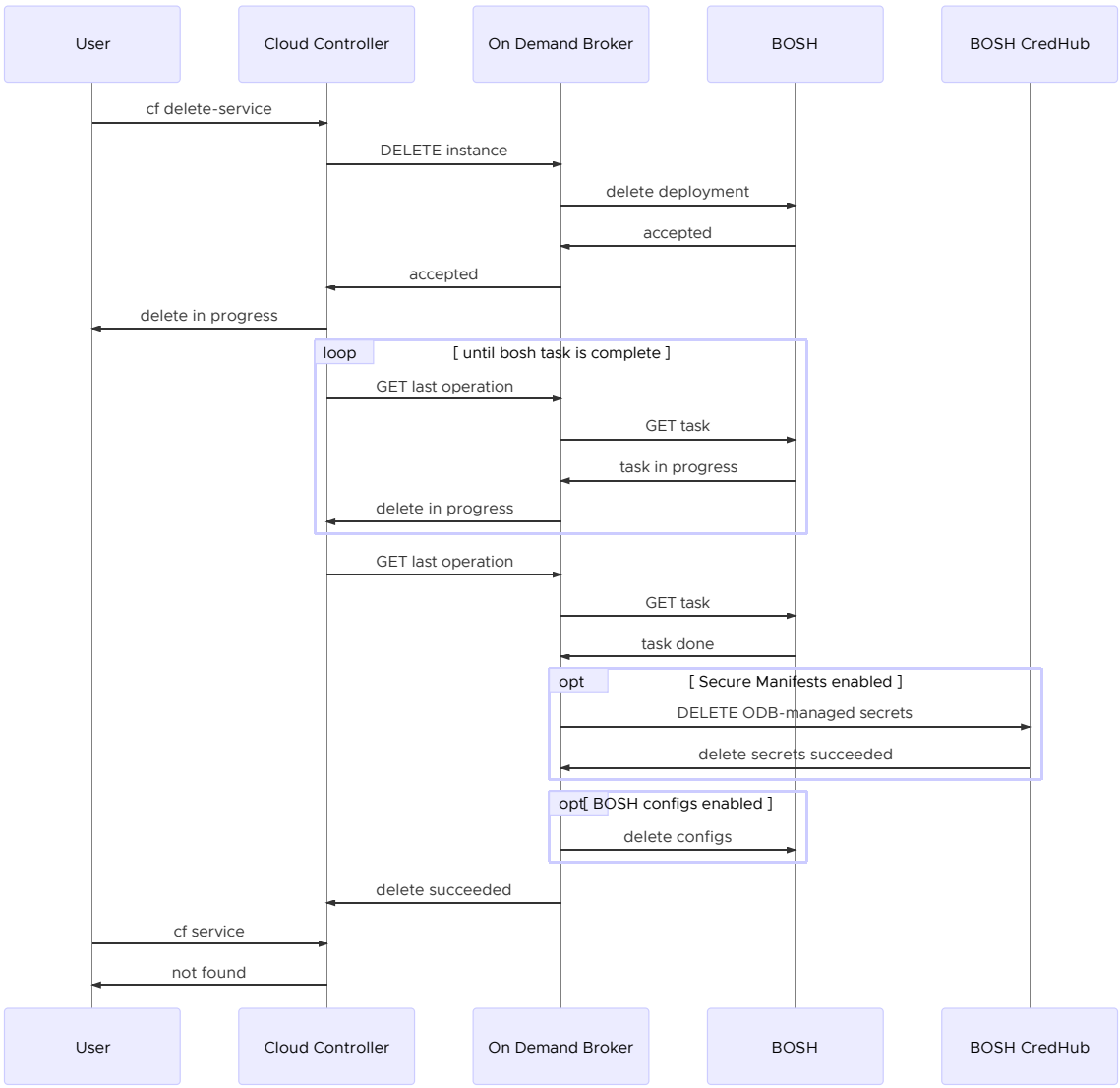
- Delete a Service Instance
- Delete a Service Instance with Pre-Delete Errands
- Delete All Service Instances
- Delete All Service Instances and Deregister Broker

Delete a Service Instance

To delete a service instance, users run the `cf delete-service` command. For more information about this command, see Delete a Service Instance.

The service adapter is not invoked in the delete service workflow.

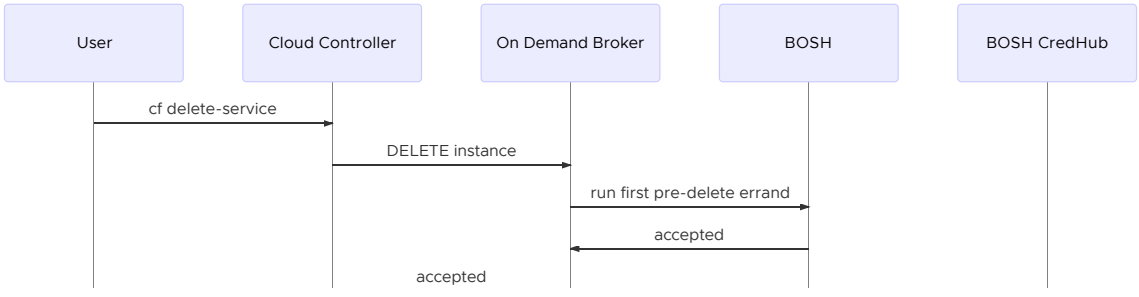
The sequence diagram below shows the workflow for deleting service instances.

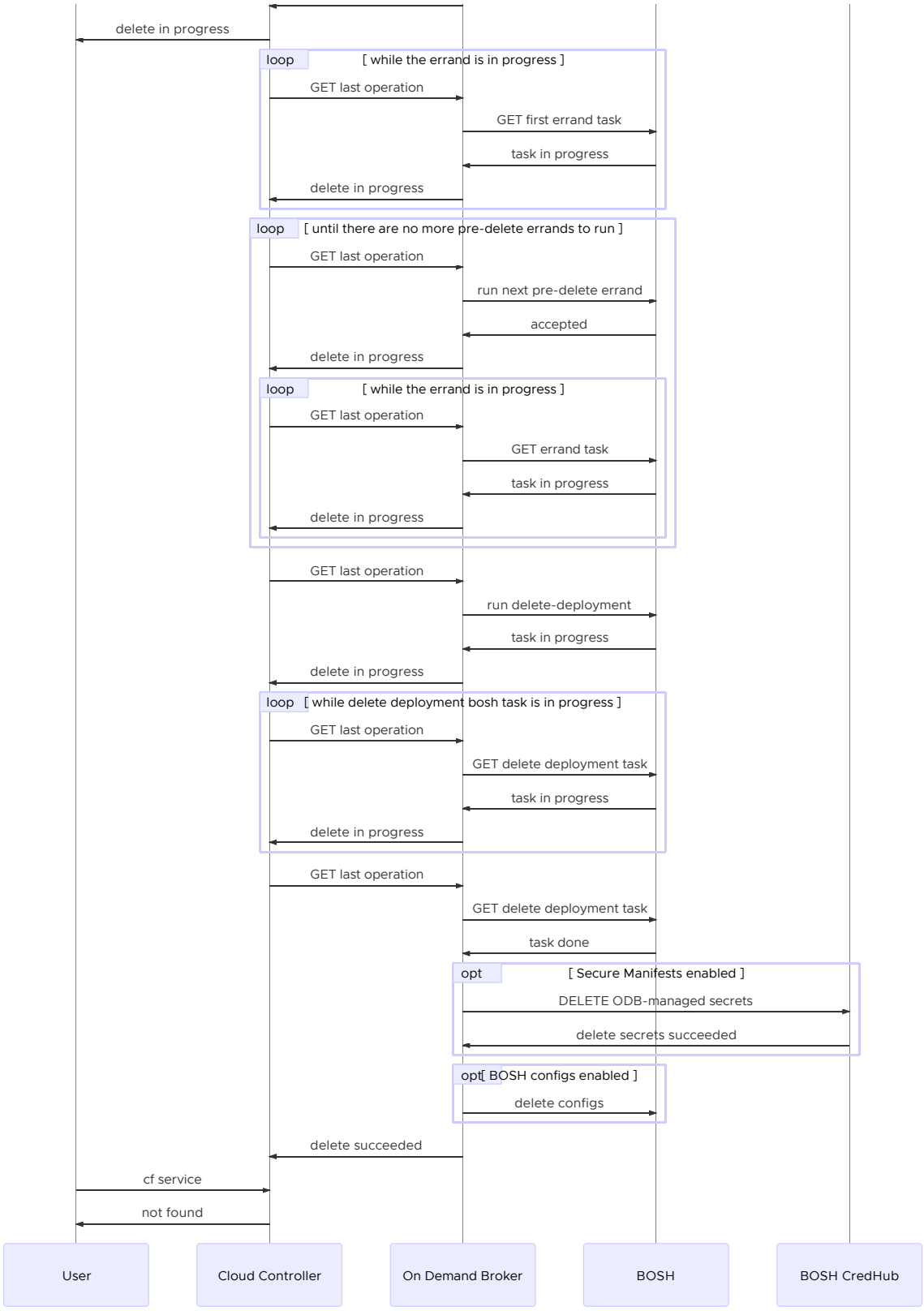


Delete a Service Instance with Pre-Delete Errands

If a user runs the `cf delete-service` command with pre-delete errands configured for the deployment, ODB does not report success to Cloud Foundry until all pre-delete errands complete and the deployment is deleted. For more information about pre-delete errands, see Service Instance Lifecycle Errands.

The sequence diagram below shows the workflow for deleting service instances with pre-delete errands configured.

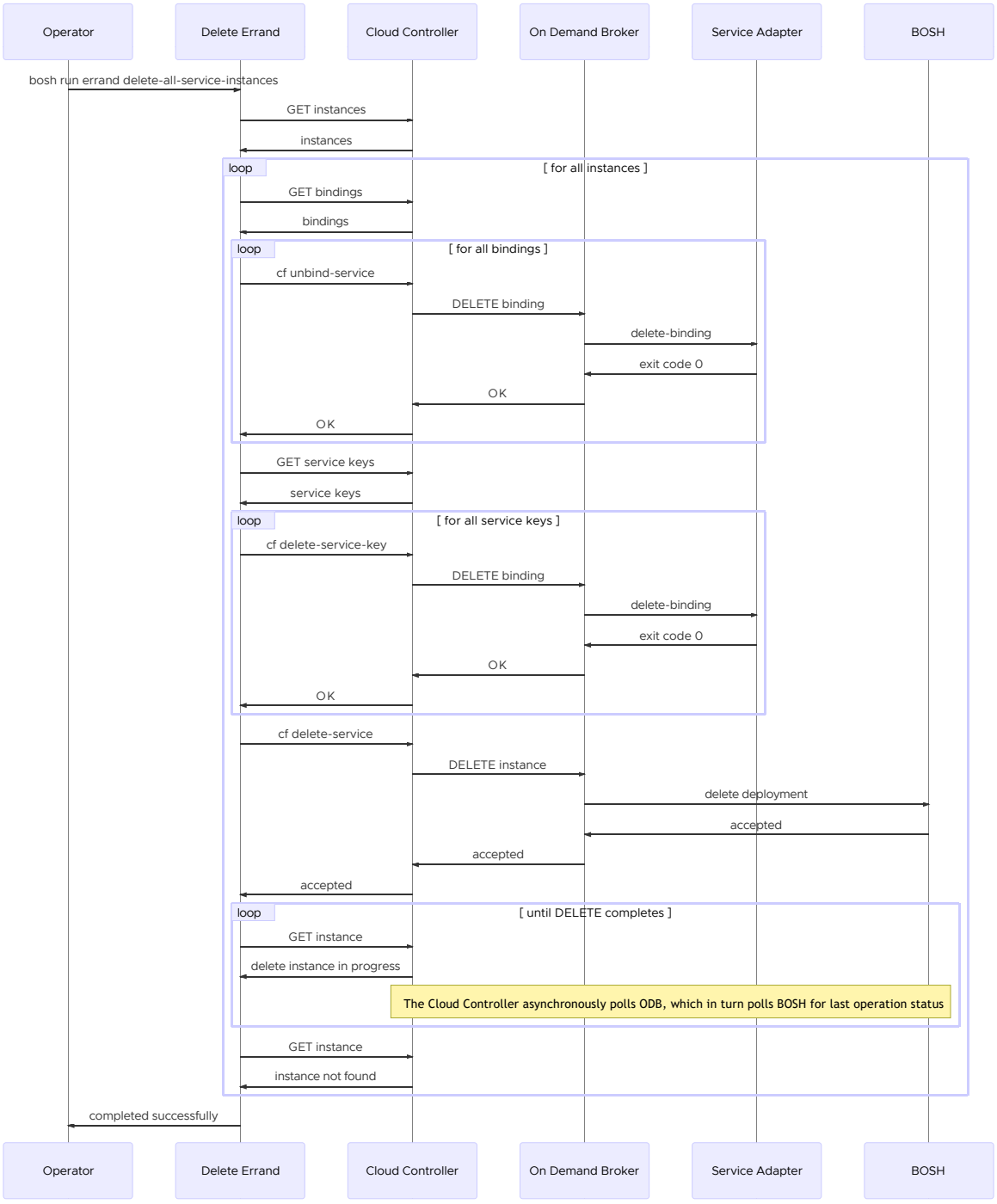




Delete All Service Instances

ODB provides the BOSH errand `delete-all-service-instances`. This errand deletes all service instances managed by the broker. For how to use this errand, see Delete All Service Instances in the Operator Guide.

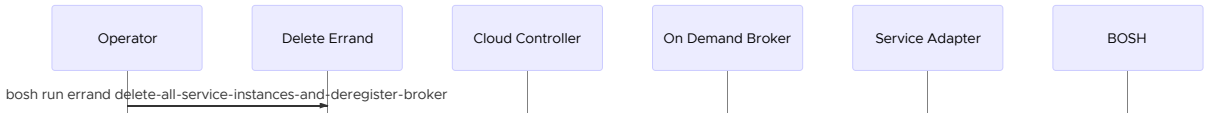
The sequence diagram below shows the workflow for deleting all service instances.

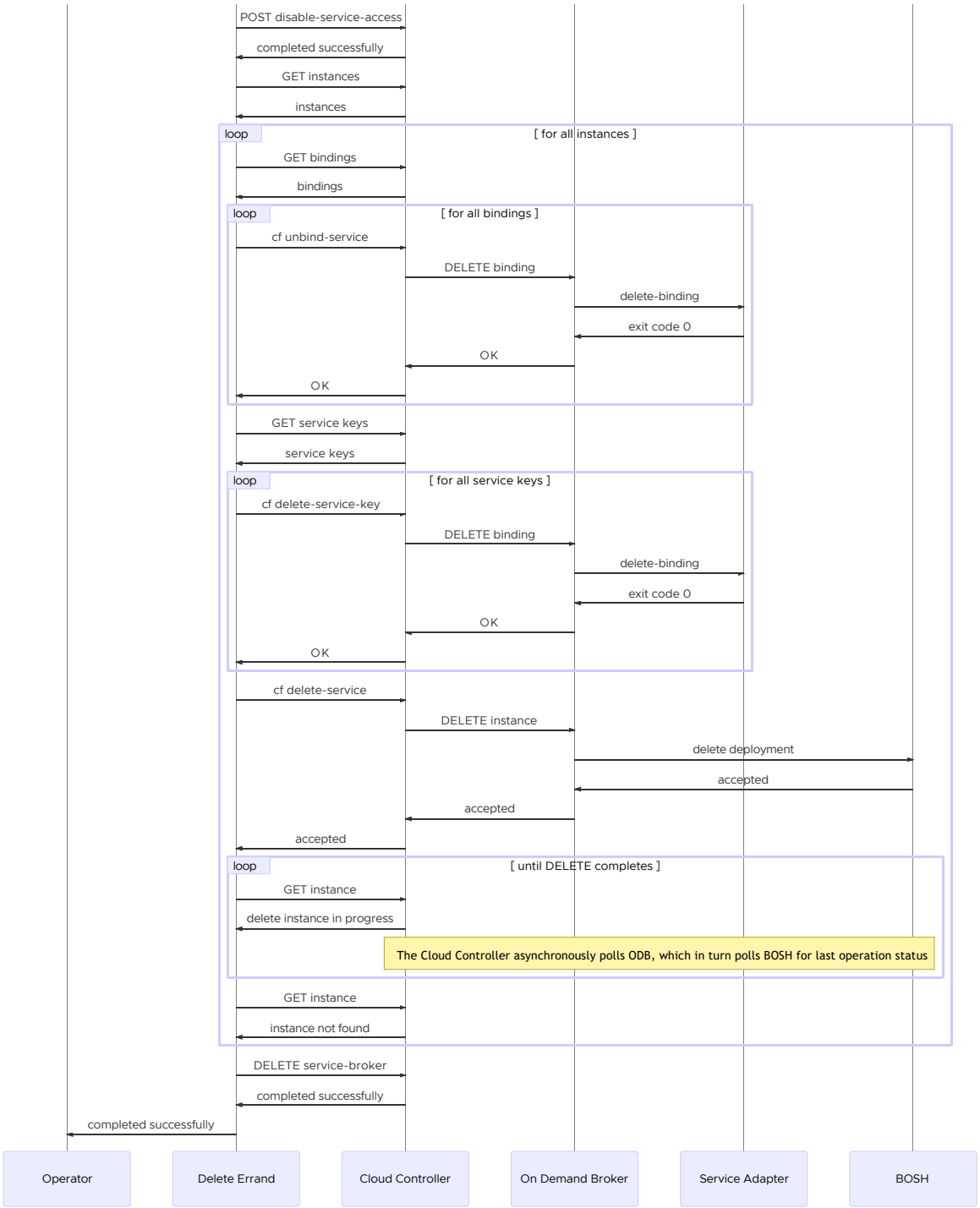


Delete All Service Instances and Deregister Broker

ODB provides the BOSH errand `delete-all-service-instances-and-deregister-broker`. This errand deletes all service instances managed by the broker and deregisters the broker from Cloud Foundry. For how to use this errand, see Delete All Service Instances and Deregister Broker in the Operator Guide.

The sequence diagram below shows the workflow for deleting all service instances and deregistering the broker.





Create a pull request or raise an issue on the source for this page in GitHub