Application Live View v1.0 Documentation

Application Live View for VMware Tanzu 1.0

Contents

Application Live View for VMware Tanzu

Note: Starting with the v1.2 release, you can find the Application Live View documentation in the Tanzu Application Platform documentation.

Application Live View is a lightweight insights and troubleshooting tool that helps application developers and application operators to look inside running applications. It is based on the concept of Spring Boot Actuators.

The fundamental idea is that the application provides information from inside the running processes via endpoints (in our case, HTTP endpoints). Application Live View uses those endpoints to get the data from the application and to interact with it.

Value Proposition

Application Live View is a diagnostic tool for developers to manage and drill into run-time characteristics of containerized applications. In addition, it provides a Kubernetes-native feel for developers to easily manage their applications in a Kubernetes environment.

Intended Audience

This information is intended for developers and operators to visualize actuator information of their running applications on Application Live View for VMware Tanzu. This information is written for developers to monitor and troubleshoot applications in development, staging, and production environments. This information is also intended for application operators to deploy and administer containerized applications in a K8s environment.

Supported Application Platforms

Application Live View can be extended to support multiple application platforms, including, but not limited to, Spring Boot and Steeltoe. Developers can use plugins to integrate their existing polyglot applications.

Multi-Cloud Compatibility

Using Tanzu platform, Application Live View can be integrated to monitor applications running across on-premises, public clouds, and edge. The platform provides a centralized view to manage applications across cloud environments, thus accelerating developer productivity and reducing time-to-market.

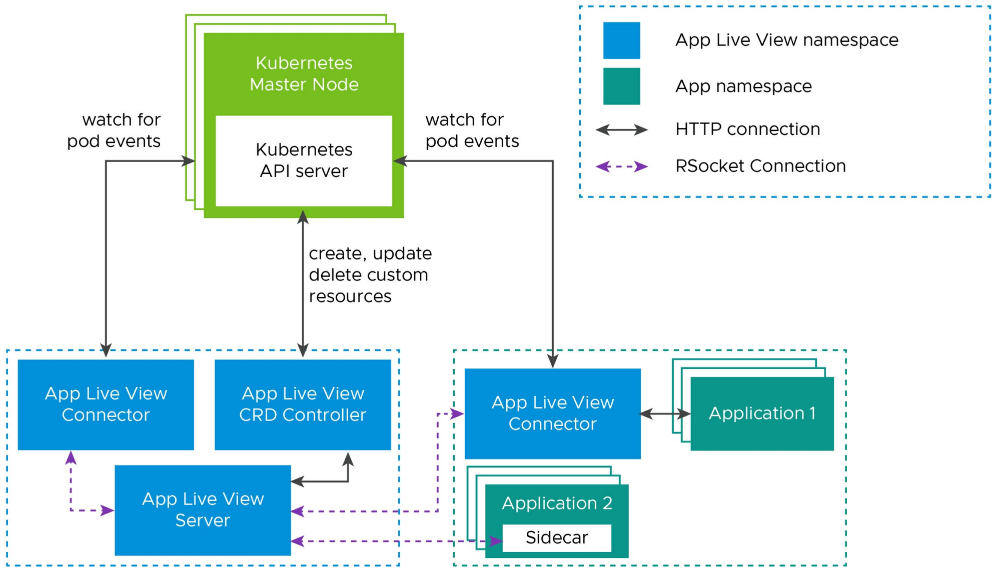
Deployment

There are two modes of deployment on registering applications with the Application Live View running on K8s cluster:

- **Connector**: A component responsible for discovering multiple applications running on K8s cluster
- **Sidecar**: A proxy component that is started alongside a single application (inside the same pod) running on K8s cluster

Architecture

This topic describes the architectural view of Application Live View and its components. This system can be deployed on a Kubernetes stack and can monitor containerized applications on hosted cloud platforms or on-premises.



Component Overview

Application Live View includes the following components as shown in the diagram above:

- **Application Live View Server**

  Application Live View Server is the central server component that contains a list of registered applications. It is responsible for proxying the request to fetch the actuator information related to the application.

- **Application Live View Connector**

  Application Live View Connector is the component responsible for discovering the application pods running on the Kubernetes cluster and register the instances to the Application Live View Server for it to be observed. The Application Live View Connector is also responsible for proxying the actuator queries to the application pods running in the Kubernetes cluster.

  Application Live View Connector can be deployed in 2 modes:

  - `Cluster access`: Application Live View Connector can be deployed as Kubernetes DaemonSet to discover applications across all the namespaces running in a worker node of a Kubernetes cluster. This is the default mode of Application Live View Connector.

  - `Namespace scoped`: Application Live View Connector can be deployed as Kubernetes Deployment to discover applications running within a namespace across worker nodes of Kubernetes cluster.

- **Sidecar**

  The Sidecar is run alongside the running application and is responsible for registering the application to Application Live View backend. Each application has a sidecar associated with it which proxies the actuator endpoint data of the application to the Application Live View Server.

- **Application Live View CRD Controller**

  Application Live View Custom Resource Definition Controller defines custom resources that return a list of registered application instances in Application Live View. It also returns metric metadata (cpu, memory) associated with the instance. The Kubernetes API server handles the storage of custom resource using etcd.

- **Application Live View Convention Webhook**

  This component provides a webhook handler for the Tanzu Convention Controller. The webhook handler is registered with Tanzu Convention Controller. What it does is detect supply-chain workloads running a Spring Boot. Such workloads are annoted automatically to enable monitoring by Application Live View.

Design flow

As illustrated in the architecture diagram, the App Live View namespace contains all the Application Live View components and the Apps namespace contains all the applications to be registered with Application Live View Server.

The applications run by the user are registered with Application Live View Server via Application Live View Connector or Sidecar.

Application Live View Connector which is a lean model uses specific labels to discover applications across cluster or namespace. Application Live View Connector talks to K8 API server asking for events for pod creation and termination and then filters out the events to find pod of interest (through labels). Once identified, then Application Live View connector will register those filtered application instances with Application Live View server. Application Live View server will proxy the call to the connector for querying actuator endpoint information.

In contrast, the Sidecar shares the pod alongside a single application and registers the application with Application Live View Server. Application Live View server will proxy the call to the sidecar for querying actuator endpoint information.

The Application Live View CRD Controller fetches the list of application instances registered with Application Live View Server and registers them as custom resources with Kubernetes API server. The Application Live View CRD Controller listens to events from the Application Live View server and creates/updates/deletes custom resources in Kubernetes server.

The Application Live View Server fetches the actuator data of the application by proxying the request to Application Live View Connector or Sidecar via RSocket connection. The Application Live View CRD Controller fetches events from Application Live View Server via HTTP connection.

Installing Application Live View

This topic describes how to install Application Live View for VMware Tanzu. You must install a Kubernetes cluster on a cloud platform provider, install command line tools, configure your cluster, and download Application Live View before installing. You install Application Live View on a Kubernetes cluster.

Application Live View installs two packages for `full` and `dev` profiles:

- Application Live View Package (run.appliveview.tanzu.vmware.com): This contains Application Live View Backend and Connector components
- Application Live View Conventions Package (build.appliveview.tanzu.vmware.com): This contains Application Live View Convention Service only

Prerequisites

The following prerequisites are required to install Application Live View:

- Kubernetes v1.20 or later
- secretgen-controller v0.5.0+
- Cert Manager v1.5.3 installed in cluster
- Tanzu Convention Controller installed in cluster
- Kapp-controller v0.24.0 or later To download kapp-controller, see the Install in the Carvel documentation.
- Command line tools. The following command line tools are required:
  - kubectl (v1.17 or later)
  - kapp (v0.37.0 or later)
  - ytt (v0.34.0 or later)
  - imgpkg (v0.14.0 or later)
  - kbld (v0.30.0 or later)
- Tanzu Cli should be installed and package plugin enabled

The steps for new users are:

1. Create a Tanzu Network account to download Tanzu Application Platform packages.
2. Visit our Application Live View for VMware Tanzu product page on Tanzu Network while logged in and confirm that you can see `Release 1.0.2`. If prompted, also accept the EULA.

To access the Application Live View UI, see Install Tanzu Application Platform GUI.

Verify the Kubernetes Cluster configuration

Run the following commands to verify the cluster configuration:

```
kubectl config current-context
kubectl cluster-info
```

Application Live View installation bundle

Perform the following procedures to install the Application Live View installation bundle.

Download the Application Live View installation bundle

Note: If you have not accepted the EULA for the Application Live View release you are about to install, then you should log in to the Tanzu Network and then go to App Live View for VMware Tanzu product page. On the page for the latest release there should be a EULA to accept, unless you already have accepted it.

You need to provide pull secrets for Tanzu Network as follows:

```
kubectl apply -f- << EOF
---
apiVersion: v1
kind: Secret
metadata:
  name: reg-creds
type: kubernetes.io/dockerconfigjson  # needs to be this type
stringData:
  .dockerconfigjson: |
    {
      "auths": {
        "registry.tanzu.vmware.com": {
          "username": "${TANZUNET_USER?:Required}",
          "password": "${TANZUNET_PASSWORD?:Required}",
          "auth": ""
        }
      }
    }

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: reg-creds
  namespace: secretgen
spec:
  toNamespaces:
  - "*"  # star means export is available for all namespaces
EOF
```

Run the following command and provide your username and password at the prompts:

```
docker login registry.tanzu.vmware.com
```

Install the Application Live View installation bundle

Now, pull the Application Live View installation bundle:

```
imgpkg pull -b registry.tanzu.vmware.com/app-live-view/application-live-view-install-b
undle:1.0.2 \
  -o /tmp/application-live-view-install-bundle
```

Deploy the Application Live View installation bundle

Use the following command to deploy the bundle:

```
ytt -f /tmp/application-live-view-install-bundle/config -f /tmp/application-live-view-
install-bundle/values.yaml \
| kbld -f /tmp/application-live-view-install-bundle/.imgpkg/images.yml -f- \
| kapp deploy -y -a application-live-view -f-
```

Note

: The Application Live View components (backend and connector) are deployed in `app-live-view` namespace by default.

Verify the Application Live View components

1. List the resources deployed in the app-live-view namespace:

   ```
   kubectl get -n app-live-view service,deploy,pod
   ```

   You should see something like this:

   ```
   NAME                                 TYPE        CLUSTER-IP     EXTERNAL-IP   P
   ORT(S)     AGE
   service/application-live-view-5112   ClusterIP   10.96.24.252   <none>        8
   0/TCP      2m56s
   service/application-live-view-7000   ClusterIP   10.96.204.81   <none>        7
   000/TCP    2m56s

   NAME                                           READY   UP-TO-DATE   AVAILABLE
    AGE
   deployment.apps/application-live-view-server   1/1     1            1
    2m56s

   NAME                                                READY   STATUS    RESTARTS
     AGE
   pod/application-live-view-connector-8tb6w           1/1     Running   0
     2m56s
   pod/application-live-view-server-567c978986-45kz2   1/1     Running   0
     2m56s
   ```

Install the Application Live View Conventions installation bundle

Perform the following procedures to install the Application Live View Conventions installation bundle.

Install the Application Live View Conventions installation bundle

Now, pull the Application Live View Conventions installation bundle:

```
imgpkg pull -b registry.tanzu.vmware.com/app-live-view/application-live-view-conventio
ns-bundle:1.0.2 \
  -o /tmp/application-live-view-conventions
```

Deploy the Application Live View Conventions installation bundle

Use the following command to deploy the bundle:

```
ytt -f /tmp/application-live-view-conventions/config -f /tmp/application-live-view-con
ventions/values.yaml \
| kbld -f /tmp/application-live-view-conventions/.imgpkg/images.yml -f- \
| kapp deploy -y -a application-live-view-conventions -f-
```

Note: The Application Live View Convention server is deployed in `alv-convention` namespace by default.

Verify the Application Live View Convention component

1. List the resources deployed in the `alv-convention` namespace:

   ```
   kubectl get -n alv-convention service,deploy,pod
   ```

2. Verify that the output is similar to this:

   ```
   NAME                           TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)
    AGE
   service/appliveview-webhook    ClusterIP   10.96.25.27   <none>        443/TCP
    11m

   NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
   deployment.apps/appliveview-webhook   1/1     1            1           11m
   ```

```
NAME                                  READY    STATUS     RESTARTS    AGE
pod/appliveview-webhook-5ff9759b6b-9pnkb   1/1      Running    0           11m
```

You can also enter your registry credentials as arguments to the `ytt` command without entering credentials in `values.yaml`. Add `registry.username` and `registry.password` arguments with your credentials as shown below:

```
ytt -f /tmp/application-live-view-install-bundle/config -f /tmp/application-live-view-install-bundle/values.yaml -v registry.server='registry.tanzu.vmware.com' -v registry.username='your username' -v registry.password='your password' \
| kbld -f /tmp/application-live-view-install-bundle/.imgpkg/images.yml -f- \
| kapp deploy -y -a application-live-view -f-
```

Application Live View is a component of Tanzu Application Platform. For how to install Tanzu Application Platform, see the Tanzu Application Platform documentation. The Application Live View UI plugin is part of Tanzu Application Platform GUI. To access the Application Live View UI, see the Application Live View in Tanzu Application Platform GUI documentation.

Uninstalling Application Live View for VMware Tanzu

This topic describes how to uninstall Application Live View for VMware Tanzu..

To uninstall the Application Live View and Application Live View Convention Server, run:

```
kapp list -A    ## Lists all the applications
kapp delete -n app-live-view -a application-live-view
kapp delete -n alv-convention -a application-live-view-conventions
kubectl delete ns app-live-view
kubectl delete ns alv-convention
```

Configuring an Application

This topic describes configuring an application from a developer perspective.

1. Convention Server
2. Spring Boot Enablement
3. Configuring Application Actuator Endpoints
4. Configuring Application for Connector
5. Scaling Applications or Knative Services

Convention Server

Application Live View Convention Server provides a Webhook handler for Convention Service for VMware Tanzu

Role of Application Live View Convention Server

The Application Live View Webhook works in conjunction with core Convention Service. It enhances Tanzu PodIntents with metadata such as labels, annotations or application properties. This metadata allows the Application Live View Connector to discover application instances to register with Application Live View Server.

The Webhook handler recognizes PodIntents for running Spring Boot apps and adds the following metadata labels and environment properties:

- `tanzu.app.live.view` (Enables the connector to observe application pod)
- `tanzu.app.live.view.application.name` (Identifies the application name to be used by Application Live View)
- `tanzu.app.live.view.application.flavours` (Exposes the framework of the application)
- `management.endpoints.web.exposure.include` (Exposes actuator endpoints of the application)
- `management.endpoint.health.show-details` (Show the health details)

Description of metadata labels are listed below:

Metadata	Default	Type	Description
`tanzu.app.live.view`	true	Label	On deployment of a Workload in TAP, this label is set to true as default across the supply chain
`tanzu.app.live.view.application.name`	spring-boot-app	Label	On deployment of a Workload in TAP, this label is set to spring-boot-app if the container image metadata doesn't contain app name, otherwise the label is set to the app name from container image metadata
`tanzu.app.live.view.application.flavours`	spring-boot	Label	On deployment of a Workload in TAP, this label is set to spring-boot as default across the supply chain
`management.endpoints.web.exposure.include`	*	Environment Property	The user provided environment property takes precedence over default value set by Application Live View Convention Server

Metadata	Default	Type	Description
`management.endpoint.health.show-details`	always	Environment Property	The user provided environment property takes precedence over default value set by Application Live View Convention Server

On deployment of a Workload from a TAP perspective, the above labels can be overriden in the `Workload` yaml. However, if the users intend to override `management.endpoints.web.exposure.include` or `management.endpoint.health.show-details`, the environment properties can be overriden in application.properties or application.yml in the Spring Boot Application before deploying the Workload in TAP. The environment properties updated by the users in their application would take precedence over the defaults set by Application Live View Convention Server.

Verify the applied labels and annotations

Assuming the name of the deployed Workload is 'tanzu-java-web-app', we can verify the applied labels and annotations using the command below:

```
kubectl get podintents.conventions.apps.tanzu.vmware.com tanzu-java-web-app -oyaml
```

```
apiVersion: conventions.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2021-11-10T10:19:38Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-appweb
    carto.run/cluster-supply-chain-name: source-to-url
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-java-web-app
    carto.run/workload-namespace: default
  name: tanzu-java-web-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-java-web-app
    uid: 998ab107-c232-4dcf-a4b2-1d499b7709c6
  resourceVersion: "4502417"
  uid: 92c65a88-5beb-4405-b659-3b78834df125
spec:
  serviceAccountName: service-account
  template:
    metadata:
      annotations:
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-appweb
        carto.run/workload-name: tanzu-java-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app@sha
256:db323d46a03e54948e844e7a7fced7d42b737c90b1c3a3a9bb775de9bce92c30
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: service-account
status:
  conditions:
  - lastTransitionTime: "2021-11-10T10:19:46Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-11-10T10:19:46Z"
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.5.4
        conventions.apps.tanzu.vmware.com/applied-conventions: |-
          appliveview-sample/app-live-view-connector
          appliveview-sample/app-live-view-appflavours
          appliveview-sample/app-live-view-systemproperties
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-appweb
        carto.run/workload-name: tanzu-java-web-app
        conventions.apps.tanzu.vmware.com/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: demo
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.show-details="always" -Dmanagement.endpo
ints.web.base-path="/actuator"
            -Dmanagement.endpoints.web.exposure.include="*" -Dmanagement.server.port="
8080"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app@sha
256:db323d46a03e54948e844e7a7fced7d42b737c90b1c3a3a9bb775de9bce92c30
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: service-account
```

`status.metadata.template.spec.containers.env.value` shows the list of applied environment properties by Application Live View Convention Server.

`status.metadata.template.labels` shows the list of applied labels by Application Live View Convention Server.

`status.metadata.template.annotations` shows the list of applied annotations by Application Live View Convention Server.

Enabling Spring Boot Applications for App Live View

This topic describes how to configure a Spring Boot Application to be observed by App Live View within TAP.

In order for App Live View to interact with a Spring Boot Application within TAP, the only requirement is that the user needs to add the `spring-boot-starter-actuator` module dependency as shown below:

Add the maven dependency in pom.xml as below:

```
<dependency>
```

```
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

The App Live View Convention Server takes care of setting the runtime environment properties `management.endpoints.web.exposure.include` and `management.endpoint.health.show-details` onto the PodSpec to expose all the actuator endpoints and detailed health information. Therefore, the user doesn't need to add these properties manually in application.properties or application.yml.

Configuring Application Actuator Endpoints

This topic describes how to configure the actuator endpoints for an application. By default, the actuator endpoint for an application is exposed on "/actuator".

The table below describes the actuator configuration scenarios and the associated labels to be used (this assumes that the application runs on port 8080):

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration	Sidecar Configuration
None	None	None	None	Actuators endpoints available at localhost:8080/actuator	tanzu.app.live.view.application.actuator.path=actuator, tanzu.app.live.view.application.actuator.port=8080	app.live.view.sidecar.application-actuator-path=actuator, app.live.view.sidecar.application-actuator-port=8080
/path	8082	/	None	Actuator endpoints available at localhost:8082/path	tanzu.app.live.view.application.actuator.path=path, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=path, app.live.view.sidecar.application-actuator-port=8082
/path	8082	/manage/actuator	None	Actuator endpoints available at localhost:8082/path/manage/actuator	tanzu.app.live.view.application.actuator.path=path/manage/actuator, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=path/manage/actuator, app.live.view.sidecar.application-actuator-port=8082
None	None	/	None	Actuators are deactivated to avoid conflicts	None	None

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration	Sidecar Configuration
None	None	/manage	None	Actuator endpoints available at /manage	tanzu.app.live.view.application.actuator.path=manage, tanzu.app.live.view.application.actuator.port=8080	app.live.view.sidecar.application-actuator-path=manage, app.live.view.sidecar.application-actuator-port=8080
/path	8082	None	None	Actuator endpoints available at localhost:8082/path/actuator	tanzu.app.live.view.application.actuator.path=path/actuator, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=path/actuator, app.live.view.sidecar.application-actuator-port=8082
/	8082	None	None	Actuator endpoints available at localhost:8082/actuator	tanzu.app.live.view.application.actuator.path=actuator, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=actuator, app.live.view.sidecar.application-actuator-port=8082
None	None	None	/api	Actuator endpoints available at localhost:8080/api/actuator	tanzu.app.live.view.application.actuator.path=api/actuator, tanzu.app.live.view.application.actuator.port=8080	app.live.view.sidecar.application-actuator-path=api/actuator, app.live.view.sidecar.application-actuator-port=8080
/path	8082	None	/api	Actuator endpoints available at localhost:8082/path/actuator	tanzu.app.live.view.application.actuator.path=path/actuator, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=path/actuator, app.live.view.sidecar.application-actuator-port=8082
/path	8082	/manage	/api	Actuator endpoints available at localhost:8082/path/manage	tanzu.app.live.view.application.actuator.path=path/manage, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=path/manage, app.live.view.sidecar.application-actuator-port=8082

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration	Sidecar Configuration
/path	None	/manage	/api	Actuator endpoints available at localhost:8080/api/manage	tanzu.app.live.view.application.actuator.path=api/manage, tanzu.app.live.view.application.actuator.port=8080	app.live.view.sidecar.application-actuator-path=api/manage, app.live.view.sidecar.application-actuator-port=8080
/path	None	/	/api	Actuators are deactivated to avoid conflicts	None	None
/path	8082	/	/api	Actuator endpoints available at localhost:8082/path	tanzu.app.live.view.application.actuator.path=path, tanzu.app.live.view.application.actuator.port=8082	app.live.view.sidecar.application-actuator-path=path, app.live.view.sidecar.application-actuator-port=8082
None	None	/manage	/api	Actuator endpoints available at localhost:8080/api/manage	tanzu.app.live.view.application.actuator.path=api/manage, tanzu.app.live.view.application.actuator.port=8080	app.live.view.sidecar.application-actuator-path=api/manage, app.live.view.sidecar.application-actuator-port=8080

Configuring Application for Connector

This topic describes how to configure an application from a developer perspective.

The connector component is responsible for discovering the application and registering it with Application Live View. The connector is deployed in the same namespace as the application. Labels are required to be added to the application PodSpec for the application to be discovered by the connector.

Below are the mandatory labels that are applied on the PodSpec by the Application Live View Convention Server by default:

```
tanzu.app.live.view="true"
tanzu.app.live.view.application.name="<app-name>"
```

The above `app-name` in `tanzu.app.live.view.application.name` is set to the Application Name from the image metadata if found, otherwise the Application Name is set to `spring-boot-app`.

Exclusive list of connector labels available are listed below:

Label Name	Mandatory	Type	Default	Significance
tanzu.app.live.view	true	Boolean	None	toggle to deactivate / activate pod discovery
tanzu.app.live.view.application.name	true	String	None	application name

Label Name	Mandatory	Type	Default	Significance
tanzu.app.live.view.application.port	false	Integer	8080	application port
tanzu.app.live.view.application.path	false	String	/	application context path
tanzu.app.live.view.application.actuator.port	false	Integer	8080	application actuator port
tanzu.app.live.view.application.actuator.path	false	String	/actuator	actuator context path
tanzu.app.live.view.application.protocol	false	http / https	http	protocol scheme
tanzu.app.live.view.application.actuator.health.port	false	Integer	8080	health endpoint port
tanzu.app.live.view.application.flavours	false	comma separated string	spring-boot	application flavours

The user can add new optional connector labels in the application `Workload` or override the existing mandatory labels such as `tanzu.app.live.view` and `tanzu.app.live.view.application.name`. If you wish to have you application not observed by Application Live View, you can override the existing label `tanzu.app.live.view = "false"`.

Configuring Developer Workload in TAP

Below is an example of a Spring PetClinic workload that overrides the connector label to `tanzu.app.live.view: "false"`

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    tanzu.app.live.view: "false"
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/spring-petclinic
```

Deploy the Workload

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

Verify the propagation of the label through the Supply Chain

- Verify if the Workload build is successful (make sure `SUCCEEDED` is set to True):

```
kubectl get builds
NAME                             IMAGE

       SUCCEEDED
spring-petclinic-build-1      dev.registry.tanzu.vmware.com/app-live-view/test/spring-p
etclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565dad7b5f0c5ac0c
df      True
```

- Check the PodIntent of your workload (`status.template.metadata.labels` shows the newly added label propagated through the Supply Chain):

```
kubectl get podintents.conventions.apps.tanzu.vmware.com spring-petclinic -oyaml

status:
  conditions:
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: Ready
  observedGeneration: 3
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.5.6
        conventions.apps.tanzu.vmware.com/applied-conventions: |-
          appliveview-sample/app-live-view-connector
          appliveview-sample/app-live-view-appflavours
          appliveview-sample/app-live-view-systemproperties
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/service-intent-mysql
        developer.conventions/target-containers: workload
        kapp.k14s.io/identity: v1;default/carto.run/Workload/spring-petclinic;carto.ru
n/v1alpha1
        kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workload","
metadata":{"annotations":{"autoscaling.knative.dev/minScale":"2"},"labels":{"app.kuber
netes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/workload-type":"web","ka
pp.k14s.io/app":"1638455805474051000","kapp.k14s.io/association":"v1.5a9384bd7b93ca74e
f494c4dec2caa4b","tanzu.app.live.view":"false"},"name":"spring-petclinic","namespace":
"default"},"spec":{"source":{"git":{"ref":{"branch":"main"},"url":"https://github.com/
ksankaranara-vmw/spring-petclinic"}}}}'
        kapp.k14s.io/original-diff-md5: 58e0494c51d30eb3494f7c9198986bb9
        services.conventions.apps.tanzu.vmware.com/mysql: mysql-connector-java/8.0.27
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: spring-petclinic
        conventions.apps.tanzu.vmware.com/framework: spring-boot
        kapp.k14s.io/app: "1638455805474051000"
        kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
        services.conventions.apps.tanzu.vmware.com/mysql: workload
        tanzu.app.live.view: "false"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: petclinic
```

- Check the ConfigMap created for the Application (`metadata.labels` shows the newly added label propagated through the Supply Chain)

```
kubectl describe configmap spring-petclinic
Name:         spring-petclinic
Namespace:    default
Labels:       carto.run/cluster-supply-chain-name=source-to-url
              carto.run/cluster-template-name=config-template
              carto.run/resource-name=app-config
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/workload-name=spring-petclinic
              carto.run/workload-namespace=default
Annotations:  <none>

Data
====
delivery.yml:
----
```

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
```

- Check the running Knative application pod (`labels` shows the newly added label on the Knative application pod)

```
kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | grep lab
els
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workload","meta
data":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":{"app.kubernete
s.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/workload-type":"web","kapp.k
14s.io/app":"1638455805474051000","kapp.k14s.io/association":"v1.5a9384bd7b93ca74ef494
c4dec2caa4b","tanzu.app.live.view":"false"},"name":"spring-petclinic","namespace":"def
ault"},"spec":{"source":{"git":{"ref":{"branch":"main"},"url":"https://github.com/ksan
karanara-vmw/spring-petclinic"}}}}'
```

The connector labels can be added or overriden in the `Workload` of your Knative application.

Scaling Knative Applications in TAP

This topic describes the working of App Live View when scaling Knative applications or developer workloads in Tanzu Application Platform.

Application Live View is focused on monitoring applications for a given `live` window and does not apply to any of the applications that are scaled down to zero. The intended behavior for Knative applications is to keep track of revisions so we can rollback easily, but also scale all of the unused revision instances down to 0 to keep the resource consumption low.

We can configure the Knative application to set `autoscaling.knative.dev/minScale` to 1 so that App Live View can still observe the instance of the application. This will make sure we have atleast 1 instance of the latest revision while still scaling down the older instances down.

We can configure any application in TAP using `Workload` resource. To scale a Knative application, the user needs to add an annotation `autoscaling.knative.dev/minScale` in the `Workload` and set to a desired value. In order for App Live View to observe an application and have atleast 1 instance of the latest revision, make sure to set the `autoscaling.knative.dev/minScale = "1"`.

The annotations or labels in the `Workload` get propagated through the TAP supply chain as below:

Workload -> PodIntent -> ConfigMap -> Push Config to repository/registry -> git-repository/imagerepository picks the Config from repository/registry -> kapp-ctrl deploys and knative runs the config -> final pod running on the Kubernetes cluster

Configuring Developer Workload in TAP

Below is an example `Workload` that adds the annotation `autoscaling.knative.dev/minScale = "1"` to set the minimum scale for spring-petclinic application:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
```

```
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/spring-petclinic
```

Deploy the Workload

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

Verify the propagation of the annotation through the Supply Chain

- Verify if the Workload build is successful (make sure `SUCCEEDED` is set to True):

```
kubectl get builds
NAME                          IMAGE

       SUCCEEDED
spring-petclinic-build-1     dev.registry.tanzu.vmware.com/app-live-view/test/spring-p
etclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565dad7b5f0c5ac0c
df     True
```

- Check the PodIntent of your workload (`status.template.metadata.annotations` shows the newly added annotation propagated through the Supply Chain):

```
kubectl get podintents.conventions.apps.tanzu.vmware.com spring-petclinic -oyaml

status:
  conditions:
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: Ready
  observedGeneration: 3
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
```

- Check the ConfigMap created for the Application (`spec.template.metadata.annotations` shows the newly added annotation propagated through the Supply Chain)

```
kubectl describe configmap spring-petclinic
Name:         spring-petclinic
Namespace:    default
Labels:       carto.run/cluster-supply-chain-name=source-to-url
              carto.run/cluster-template-name=config-template
              carto.run/resource-name=app-config
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/workload-name=spring-petclinic
              carto.run/workload-namespace=default
Annotations:  <none>

Data
====
delivery.yml:
----
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
```

- Check the running Knative application pod (`annotations` shows the newly added annotation on the Knative application pod)

```
kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | grep ann
otations
  annotations:
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workload","meta
data":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":{"app.kubernete
s.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/workload-type":"web","kapp.k
14s.io/app":"1638455805474051000","kapp.k14s.io/association":"v1.5a9384bd7b93ca74ef494
c4dec2caa4b","tanzu.app.live.view":"false"},"name":"spring-petclinic","namespace":"def
ault"},"spec":{"source":{"git":{"ref":{"branch":"main"},"url":"https://github.com/ksan
karanara-vmw/spring-petclinic"}}}}'
```

Your Knative application is now set to a minimum scale of 1 so that Application Live View can observe the instance of the application.

Product Features

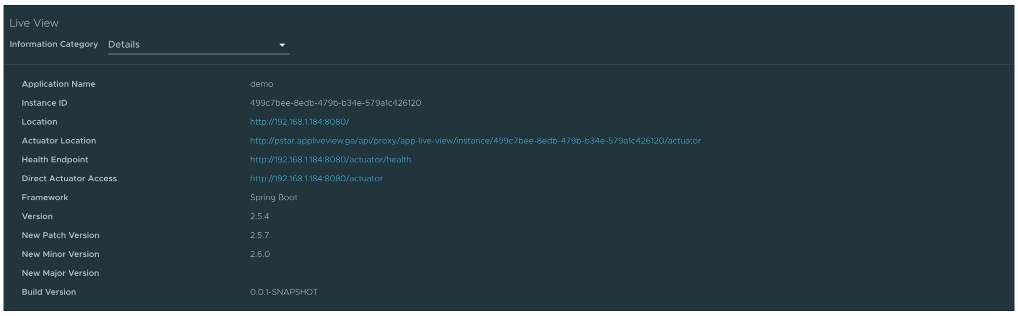
This topic explains the Application Live View UI features.

The Application Live View encompasses a sophisticated UI that provides visual insights into running applications by inspecting the application actuator information. The actuator data from the application serves as the source of truth. Application Live View provides a live view of the data from inside of the application only. Application Live View does not store any of the applications data for further analysis or historical views. This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the applications. It gives certain level of control to the users to change some parameters such as log levels, environment properties of running applications.

Application Live View UI includes the following pages and views:

Details page

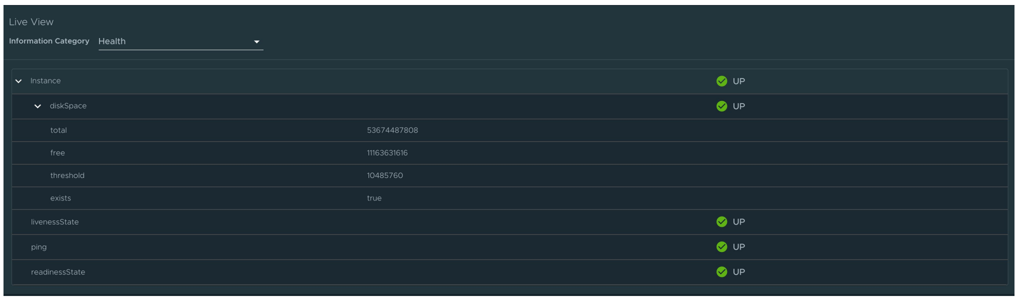
This is the default page loaded in the `Live View` section. This page gives a tabular overview containing the application name, instance id, location, actuator location, health endpoint, direct actuator access, framework , version, new patch version, new major version, build version. The user can navigate between `Information Categories` by selecting from the drop-down menu on the top right corner of the page.



Application Name	demo
Instance ID	499c7bee-8edb-479b-b34e-579a1c426120
Location	http://192.168.1.184:8080/
Actuator Location	http://pstar.appliveview.ga/api/proxy/app-live-view/instance/499c7bee-8edb-479b-b34e-579a1c426120/actuator
Health Endpoint	http://192.168.1.184:8080/actuator/health
Direct Actuator Access	http://192.168.1.184:8080/actuator
Framework	Spring Boot
Version	2.5.4
New Patch Version	2.5.7
New Minor Version	2.6.0
New Major Version	
Build Version	0.0.1-SNAPSHOT

Health page

To navigate to the health page, the user should select the `Health` option from the `Information Category` dropdown. The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application like readiness, liveness and disk space. It displays the status, details associated with each of the components.



Instance		✓ UP
diskSpace		✓ UP
total	53674487808	
free	11163631616	
threshold	10485760	
exists	true	
livenessState		✓ UP
ping		✓ UP
readinessState		✓ UP

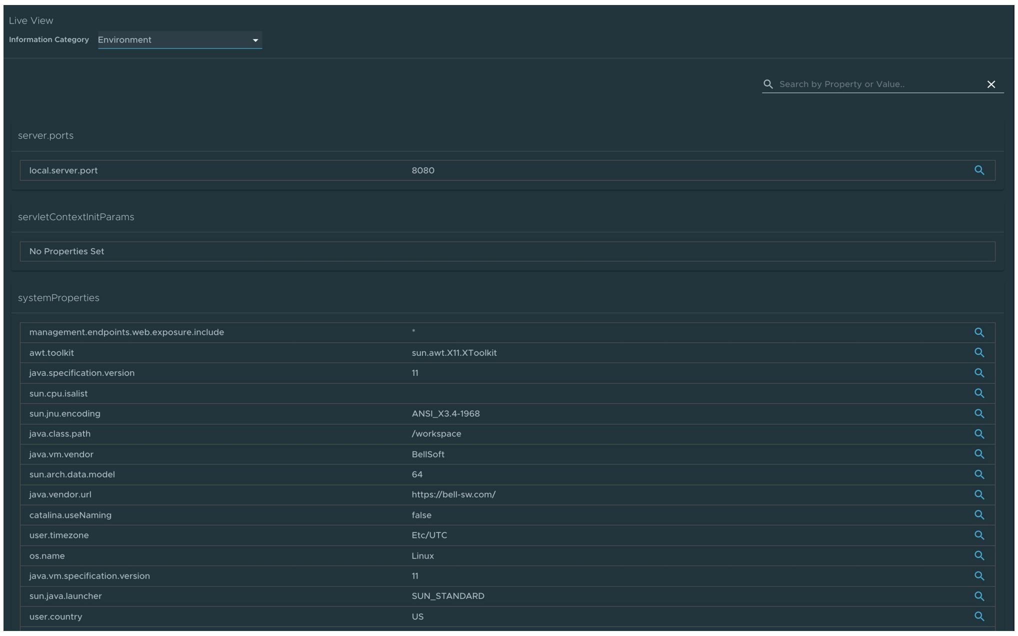
Environment page

To navigate to the Environment page, the user should select the `Environment` option from the `Information Category` dropdown. The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (like application.properties) in a Spring Boot application.

The page includes the following features:

- The UI has search feature that allows the user to search for a property or values.
- Each property has a search icon at the right corner which helps the user quickly see all the occurrences of a specific property key without manually typing in the search field. Clicking on this button trims down the page to that property name.
- The *Refresh Scope* button on the top right corner of the page is used to probe the application to refresh all the environment properties.
- The user can modify existing property by clicking on the override button in the row and editing the value. Once the value is saved, the user can see the updated property in the Applied overrides section at the top of the page.
- The *Reset* button is used to reset the environment property to the original state.
- The overridden environment variables in the *Applied Overrides* section can be edited or removed.
- The *Applied Overrides* section also allows the user to add new environment properties to the application.

NOTE: The `management.endpoint.env.post.enabled=true` has to be set in the application config properties of the application as well as a corresponding, editable Environment has to be present in the (here Spring Boot) application.



server.ports

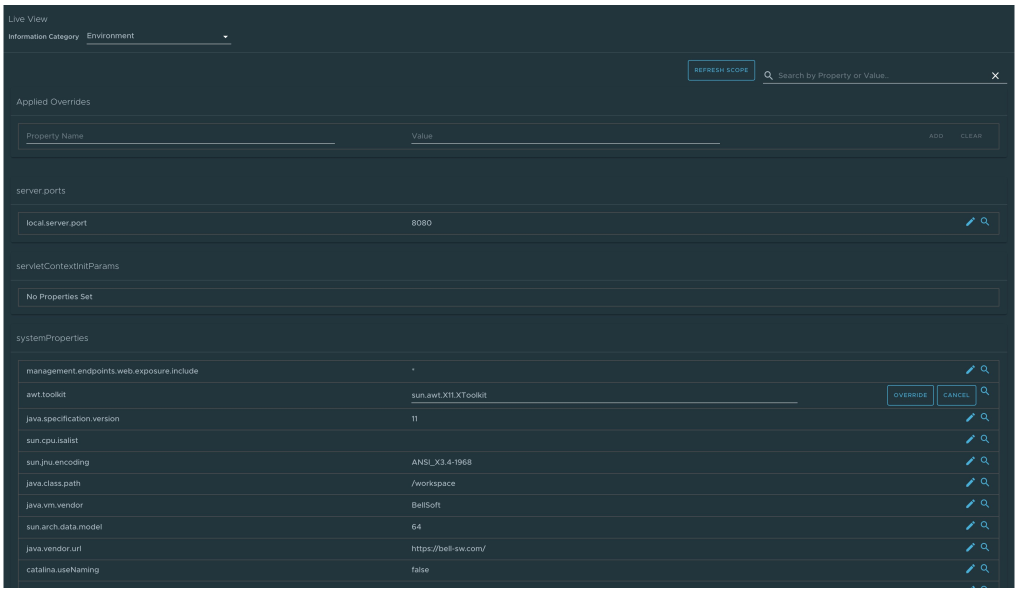
local.server.port	8080

servletContextInitParams

No Properties Set

systemProperties

management.endpoints.web.exposure.include	*
awt.toolkit	sun.awt.X11.XToolkit
java.specification.version	11
sun.cpu.isalist	
sun.jnu.encoding	ANSI_X3.4-1968
java.class.path	/workspace
java.vm.vendor	BellSoft
sun.arch.data.model	64
java.vendor.url	https://bell-sw.com/
catalina.useNaming	false
user.timezone	Etc/UTC
os.name	Linux
java.vm.specification.version	11
sun.java.launcher	SUN_STANDARD
user.country	US



Live View

Information Category: Environment

REFRESH SCOPE | Search by Property or Value..

Applied Overrides

Property Name	Value	
		ADD CLEAR

server.ports

local.server.port	8080

servletContextInitParams

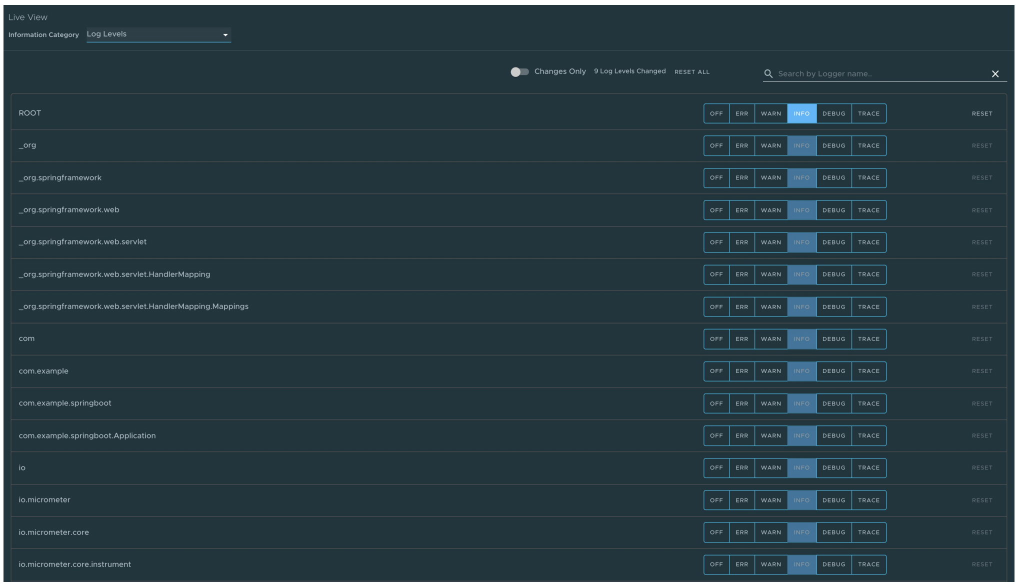
No Properties Set

systemProperties

management.endpoints.web.exposure.include	*	
awt.toolkit	sun.awt.X11.XToolkit	OVERRIDE CANCEL
java.specification.version	11	
sun.cpu.isalist		
sun.jnu.encoding	ANSI_X3.4-1968	
java.class.path	/workspace	
java.vm.vendor	BellSoft	
sun.arch.data.model	64	
java.vendor.url	https://bell-sw.com/	
catalina.useNaming	false	

Log Levels Page

To navigate to the Log Levels page, the user should select the `Log Levels` option from the `Information Category` dropdown. The log levels page provides access to the application's loggers and the configuration of their levels. The log levels such as INFO, DEBUG, TRACE can be configured real-time by the user from the UI. The user can search for a package and modify its respective log level. The user has the ability to configure the log levels at a specific class and package. They can turn off all the log levels by modifying the log level of root logger to OFF. The toggle `Changes Only` displays the changed log levels. The search feature allows the user to search by logger name. The *Reset* button resets the log levels to the original state. The *Reset All* button on top right corner of the page resets all the loggers to default state.



Live View

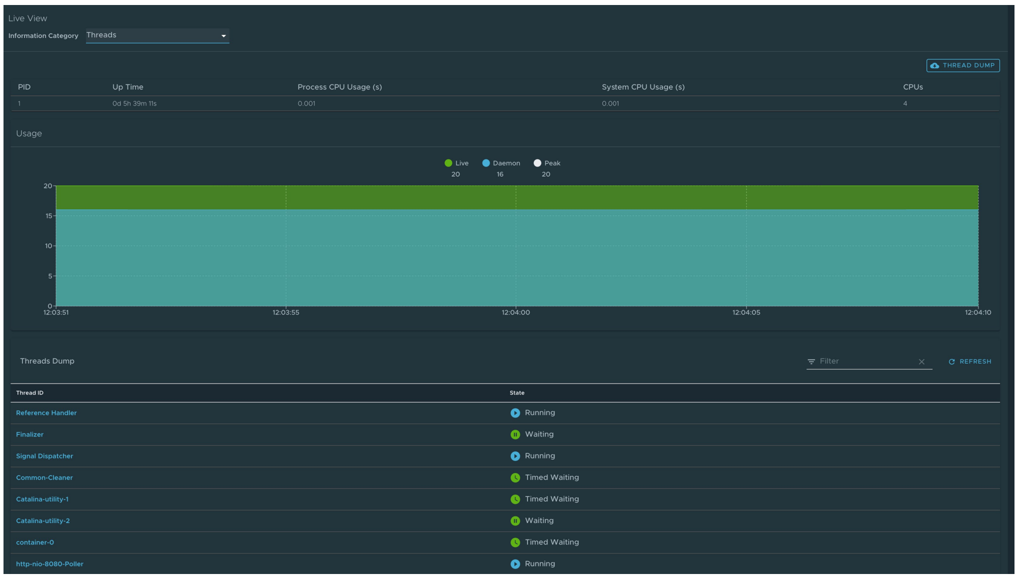
Information Category: Log Levels

Changes Only | 9 Log Levels Changed | RESET ALL | Search by Logger name..

ROOT	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
_org	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
_org.springframework	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
_org.springframework.web	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
_org.springframework.web.servlet	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
_org.springframework.web.servlet.HandlerMapping	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
_org.springframework.web.servlet.HandlerMapping.Mappings	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
com	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
com.example	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
com.example.springboot	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
com.example.springboot.Application	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
io	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
io.micrometer	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
io.micrometer.core	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET
io.micrometer.core.instrument	OFF	ERR	WARN	INFO	DEBUG	TRACE	RESET

Threads Page

To navigate to the Threads page, the user should select the `Threads` option from the `Information Category` dropdown. This page displays all details related to JVM threads and running processes of the application. This tracks live threads and daemon threads real-time. It is a snapshot of different thread states. Navigating to a thread state displays all the information about a particular thread and its stack trace. The search feature allows the user to search for threads by thread ID or state. The refresh icon refreshes to the latest state of the threads. The user can view more thread details by clicking on the Thread ID. The page also has a feature to download thread dump for analysis purposes.

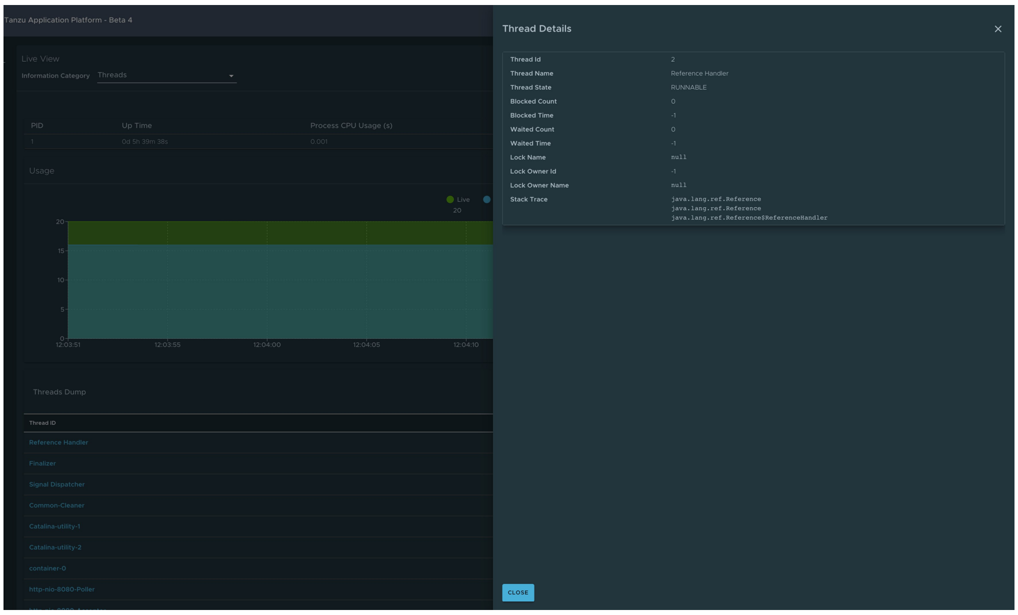

Live View

Information Category: Threads

PID	Up Time	Process CPU Usage (s)	System CPU Usage (s)	CPUs
1	0d 5h 39m 11s	0.001	0.001	4

Threads Dump

Thread ID	State
Reference Handler	Running
Finalizer	Waiting
Signal Dispatcher	Running
Common-Cleaner	Timed Waiting
Catalina-utility-1	Timed Waiting
Catalina-utility-2	Waiting
container-0	Timed Waiting
http-nio-8080-Poller	Running

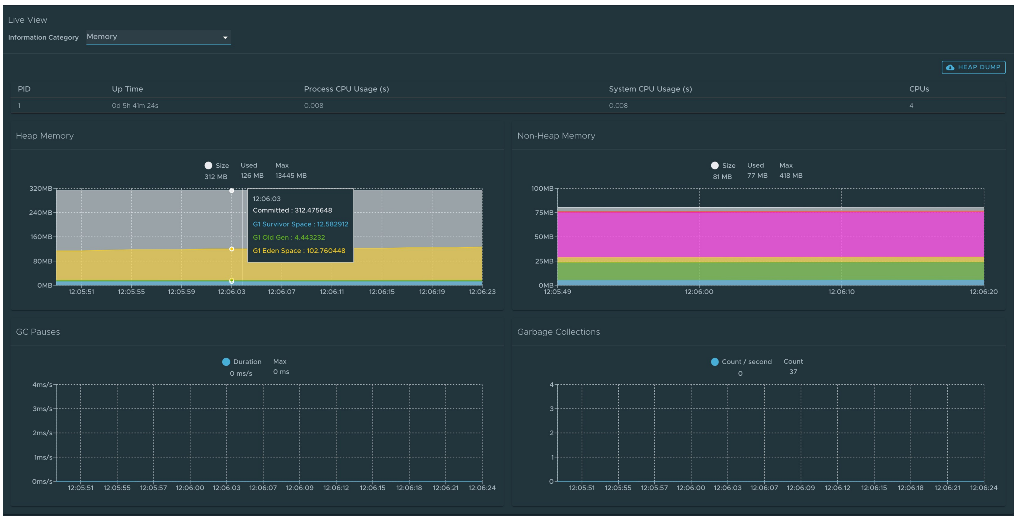


Thread Details

Thread Id	2
Thread Name	Reference Handler
Thread State	RUNNABLE
Blocked Count	0
Blocked Time	-1
Waited Count	0
Waited Time	-1
Lock Name	null
Lock Owner Id	-1
Lock Owner Name	null
Stack Trace	java.lang.ref.Reference java.lang.ref.Reference java.lang.ref.Reference$ReferenceHandler

Memory Page

To navigate to the Memory page, the user should select the `Memory` option from the `Information Category` dropdown. * The memory page highlights the memory usage inside of the JVM. It displays a graphical representation of the different memory regions within heap and non-heap memory. Please note that this visualizes data from inside of the JVM (in case of Spring Boot apps running on a JVM) and therefore provides memory insights into the application in contrast to “outside” information on the k8s pod level. * The real-time graphs displays a stacked overview of the different spaces in memory along with the total memory used and total memory size. The page contains graphs to display the GC pauses and GC events. The Heap Dump button on top right corner allows the user to download heap dump data.


Live View

Information Category: Memory

PID	Up Time	Process CPU Usage (s)	System CPU Usage (s)	CPUs
1	0d 5h 41m 24s	0.008	0.008	4

Heap Memory — Size 312 MB, Used 126 MB, Max 13445 MB

12:06:03
Committed : 312.475648
G1 Survivor Space : 12.582912
G1 Old Gen : 4.443232
G1 Eden Space : 102.760448

Non-Heap Memory — Size 81 MB, Used 77 MB, Max 418 MB

GC Pauses — Duration 0 ms/s, Max 0 ms

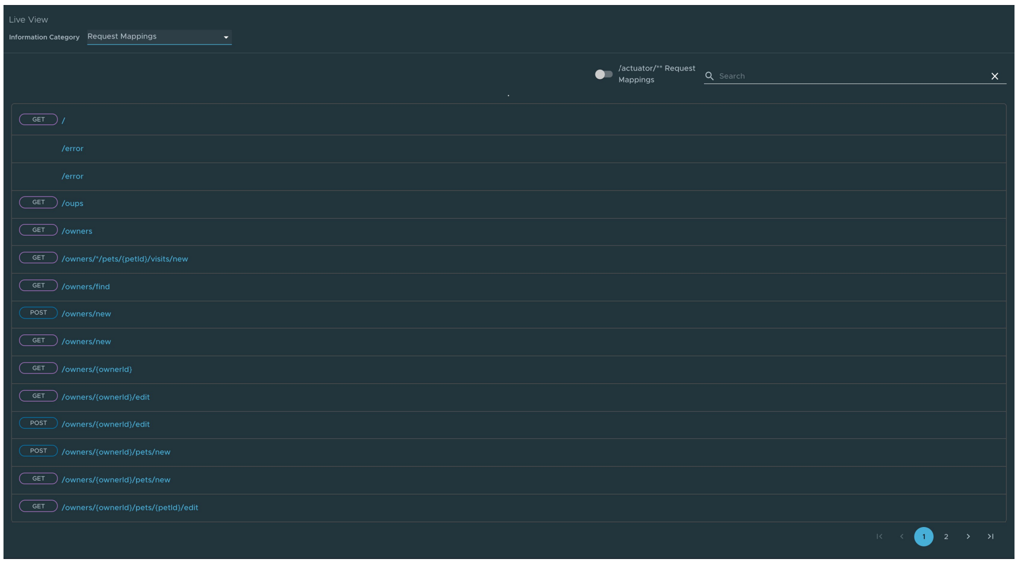
Garbage Collections — Count / second 0, Count 37

Please keep in mind that this graphical visualization happens in real-time and shows real-time data only. As mentioned at the top, the Application Live View features do not store any information. That means the graphs visualize the data over time only for as long as you stay on that page.

Request Mappings Page

To navigate to the Request Mappings page, the user should select the `Request Mappings` option from the `Information Category` dropdown. This page provides information about the application's request mappings. For each of the mapping, it displays the request handler method. The user can view more details of the request mapping such as header metadata of the application, i.e produces, consumes and HTTP method by clicking on the mapping. The search feature allows the user to search on the request mapping or the method. The toggle `/actuator/** Request Mappings` displays the actuator related mappings of the application.

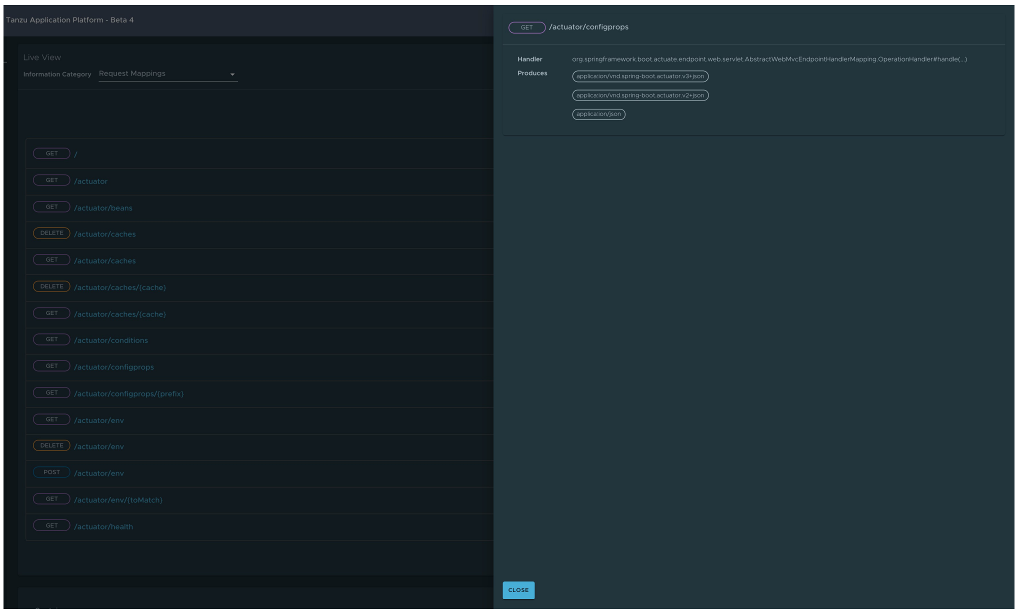
NOTE: When application actuator endpoint is exposed on management.server.port, the application does not return any actuator request mappings data in the context. In this case, a message is displayed when the actuator toggle is activated.


Live View

Information Category: Request Mappings

/actuator/** Request Mappings

GET	/
	/error
	/error
GET	/oups
GET	/owners
GET	/owners/*/pets/{petId}/visits/new
GET	/owners/find
POST	/owners/new
GET	/owners/new
GET	/owners/{ownerId}
GET	/owners/{ownerId}/edit
POST	/owners/{ownerId}/edit
POST	/owners/{ownerId}/pets/new
GET	/owners/{ownerId}/pets/new
GET	/owners/{ownerId}/pets/{petId}/edit


Tanzu Application Platform - Beta 4

Live View

Information Category: Request Mappings

GET	/
GET	/actuator
GET	/actuator/beans
DELETE	/actuator/caches
GET	/actuator/caches
DELETE	/actuator/caches/{cache}
GET	/actuator/caches/{cache}
GET	/actuator/conditions
GET	/actuator/configprops
GET	/actuator/configprops/{prefix}
GET	/actuator/env
DELETE	/actuator/env
POST	/actuator/env
GET	/actuator/env/{toMatch}
GET	/actuator/health

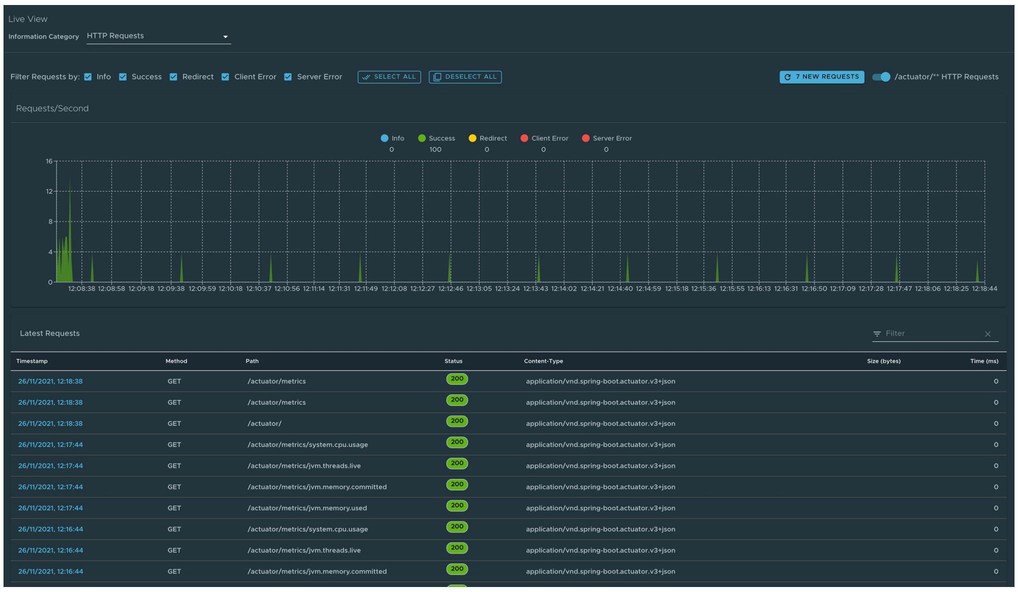
GET /actuator/configprops

Handler	org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMapping.OperationHandler#handle(...)
Produces	application/vnd.spring-boot.actuator.v3+json
	application/vnd.spring-boot.actuator.v2+json
	application/json

HTTP Requests Page

To navigate to the HTTP Requests page, the user should select the `HTTP Requests` option from the `Information Category` dropdown. The HTTP Requests page provides information about HTTP request-response exchanges to the application. The graph visualizes the requests per second indicating the response status of all the requests. The user can filter on the response statuses which include info, success, redirects, client-errors, server-errors. The trace data is captured in detail in a tabular format with metrics such as timestamp, method, path, status, content-type, length, time. The search feature on the table filters the traces based on the search field value. The user can view more details of the request such as method, headers, response of the application by clicking on the timestamp. The refresh icon above the graph loads the latest traces of the application. The toggle '/actuator/**' on the top right corner of the page displays the actuator related traces of the application.

Note: When the application actuator endpoint is exposed on management.server.port, no actuator HTTP Traces data is returned for the application. In this case, a message is displayed when the actuator toggle is activated.

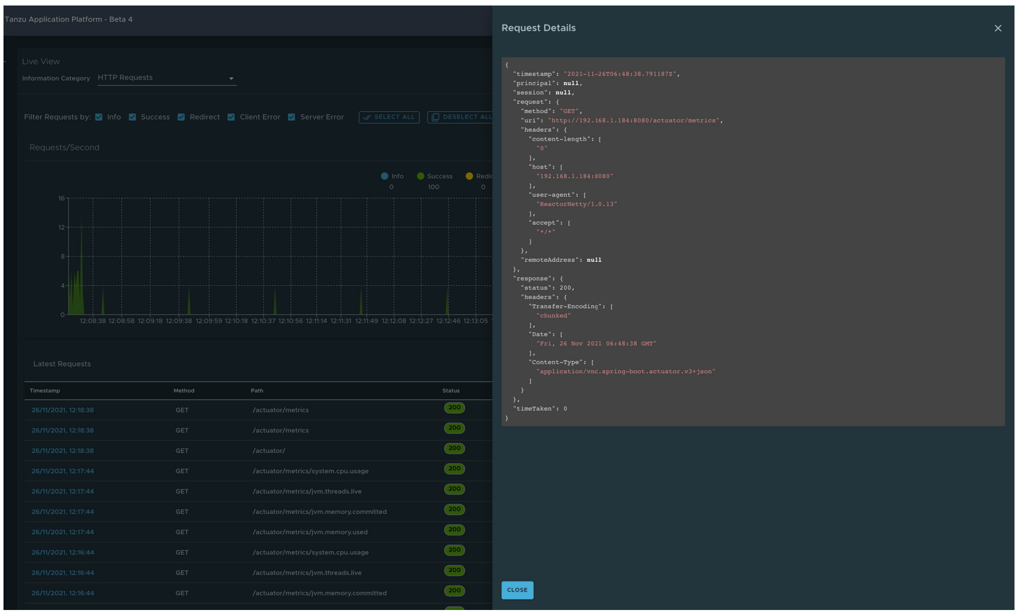

Live View

Information Category: HTTP Requests

Filter Requests by: ✓ Info ✓ Success ✓ Redirect ✓ Client Error ✓ Server Error

Latest Requests

Timestamp	Method	Path	Status	Content-Type	Size (bytes)	Time (ms)
26/11/2021, 12:18:38	GET	/actuator/metrics	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:18:38	GET	/actuator/metrics	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:18:38	GET	/actuator/	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/system.cpu.usage	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.threads.live	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.committed	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.used	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:16:44	GET	/actuator/metrics/system.cpu.usage	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.threads.live	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.memory.committed	200	application/vnd.spring-boot.actuator.v3+json		0

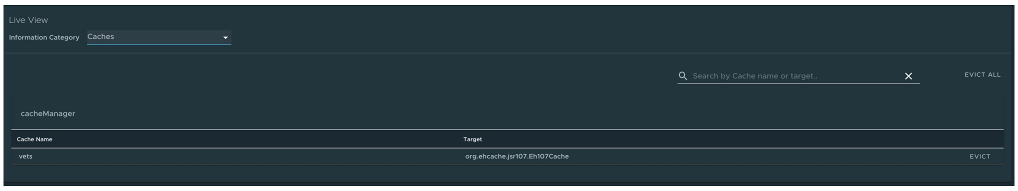


Timestamp	Method	Path	Status
26/11/2021, 12:18:38	GET	/actuator/metrics	200
26/11/2021, 12:18:38	GET	/actuator/metrics	200
26/11/2021, 12:18:38	GET	/actuator/	200
26/11/2021, 12:17:44	GET	/actuator/metrics/system.cpu.usage	200
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.threads.live	200
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.committed	200
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.used	200
26/11/2021, 12:16:44	GET	/actuator/metrics/system.cpu.usage	200
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.threads.live	200
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.memory.committed	200

```
{
  "timestamp": "2021-11-26T06:48:38.791187Z",
  "principal": null,
  "session": null,
  "request": {
    "method": "GET",
    "uri": "http://192.168.1.184:8080/actuator/metrics",
    "headers": {
      "content-length": [
        "0"
      ],
      "host": [
        "192.168.1.184:8080"
      ],
      "user-agent": [
        "ReactorNetty/1.0.13"
      ],
      "accept": [
        "*/*"
      ]
    },
    "remoteAddress": null
  },
  "response": {
    "status": 200,
    "headers": {
      "Transfer-Encoding": [
        "chunked"
      ],
      "Date": [
        "Fri, 26 Nov 2021 06:48:38 GMT"
      ],
      "Content-Type": [
        "application/vnd.spring-boot.actuator.v3+json"
      ]
    }
  },
  "timeTaken": 0
}
```

Caches Page

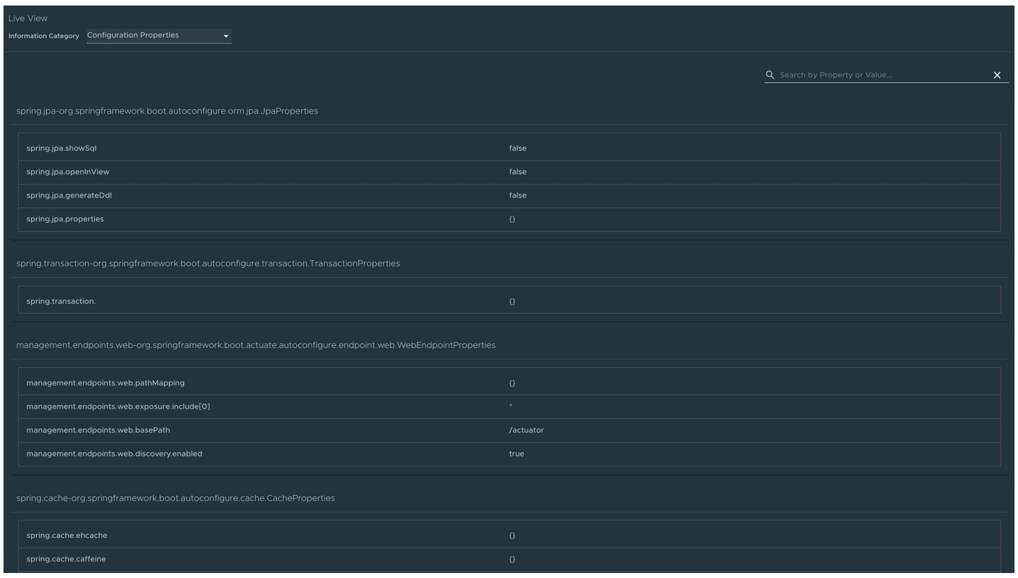
To navigate to the Caches page, the user should select the `Caches` option from the `Information Category` dropdown. The Caches page provides access to the application's caches. It gives the details of the cache managers associated with the application including the fully qualified name of the native cache. The search feature in the Caches Page allows the user to search for a specific cache/cache manager. The user has the ability to evict individual caches by clicking on the *Evict* button, which results in clearing of cache. All the caches can be evicted completely by clicking on *Evict All* button. If there are no cache managers for the application, a message is displayed `No cache managers available for the application`.



Cache Name	Target	
vets	org.ehcache.jsr107.Eh107Cache	EVICT

Configuration Properties Page

To navigate to the Configuration Properties page, the user should select the `Configuration Properties` option from the `Information Category` dropdown. The configuration properties page provides information about the configuration properties of the application. In case of Spring Boot, it displays application's @ConfigurationProperties beans. It gives a snapshot of all the beans and their associated configuration properties. The search feature allows the user to look up for property's key/value or the bean name.


Live View

Information Category: Configuration Properties

Search by Property or Value...

spring.jpa-org.springframework.boot.autoconfigure.orm.jpa.JpaProperties

spring.jpa.showSql	false
spring.jpa.openInView	false
spring.jpa.generateDdl	false
spring.jpa.properties	{}

spring.transaction-org.springframework.boot.autoconfigure.transaction.TransactionProperties

spring.transaction.	{}

management.endpoints.web-org.springframework.boot.actuate.autoconfigure.endpoint.web.WebEndpointProperties

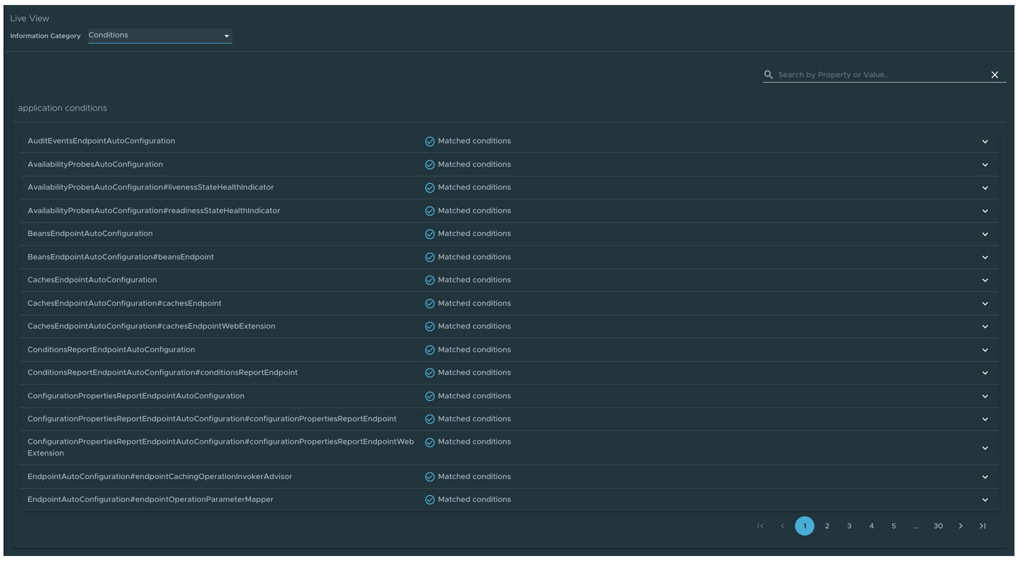
management.endpoints.web.pathMapping	{}
management.endpoints.web.exposure.include[0]	*
management.endpoints.web.basePath	/actuator
management.endpoints.web.discovery.enabled	true

spring.cache-org.springframework.boot.autoconfigure.cache.CacheProperties

spring.cache.ehcache	{}
spring.cache.caffeine	{}

Conditions Page

To navigate to the Conditions page, the user should select the `Conditions` option from the `Information Category` dropdown. The conditions evaluation report provides information about the evaluation of conditions on configuration and auto-configuration classes. In case of Spring Boot, this gives the user a clear view of all the beans configured in the application. When the user clicks on the bean name, the conditions and the reason for the conditional match is displayed. In case of not configured beans, it shows both the matched and unmatched conditions of the bean if any. In addition to this, it also displays names of unconditional auto configuration classes if any. The user can filter out on the beans and the conditions using the search feature.


Live View

Information Category: Conditions

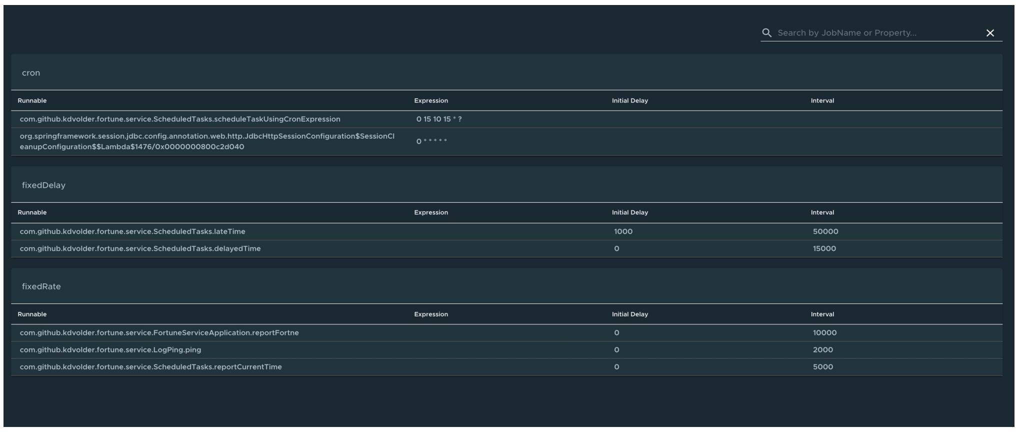
Search by Property or Value...

application conditions

AuditEventsEndpointAutoConfiguration	✓ Matched conditions
AvailabilityProbesAutoConfiguration	✓ Matched conditions
AvailabilityProbesAutoConfiguration#livenessStateHealthIndicator	✓ Matched conditions
AvailabilityProbesAutoConfiguration#readinessStateHealthIndicator	✓ Matched conditions
BeansEndpointAutoConfiguration	✓ Matched conditions
BeansEndpointAutoConfiguration#beansEndpoint	✓ Matched conditions
CachesEndpointAutoConfiguration	✓ Matched conditions
CachesEndpointAutoConfiguration#cachesEndpoint	✓ Matched conditions
CachesEndpointAutoConfiguration#cachesEndpointWebExtension	✓ Matched conditions
ConditionsReportEndpointAutoConfiguration	✓ Matched conditions
ConditionsReportEndpointAutoConfiguration#conditionsReportEndpoint	✓ Matched conditions
ConfigurationPropertiesReportEndpointAutoConfiguration	✓ Matched conditions
ConfigurationPropertiesReportEndpointAutoConfiguration#configurationPropertiesReportEndpoint	✓ Matched conditions
ConfigurationPropertiesReportEndpointAutoConfiguration#configurationPropertiesReportEndpointWebExtension	✓ Matched conditions
EndpointAutoConfiguration#endpointCachingOperationInvokerAdvisor	✓ Matched conditions
EndpointAutoConfiguration#endpointOperationParameterMapper	✓ Matched conditions

Scheduled Tasks Page

To navigate to the Scheduled Tasks page, the user should select the `Scheduled Tasks` option from the `Information Category` dropdown. The scheduled tasks page provides information about the application's scheduled tasks. It includes cron tasks, fixed delay tasks and fixed rate tasks, custom tasks and the properties associated with them. The user can search for a particular property or a task in the search bar to retrieve the task or property details.



cron

Runnable	Expression	Initial Delay	Interval
com.github.kdvolder.fortune.service.ScheduledTasks.scheduleTaskUsingCronExpression	0 15 10 15 * ?		
org.springframework.session.jdbc.config.annotation.web.http.JdbcHttpSessionConfiguration$SessionCleanupConfiguration$$Lambda$1476/0x0000000800c2d040	0 * * * * *		

fixedDelay

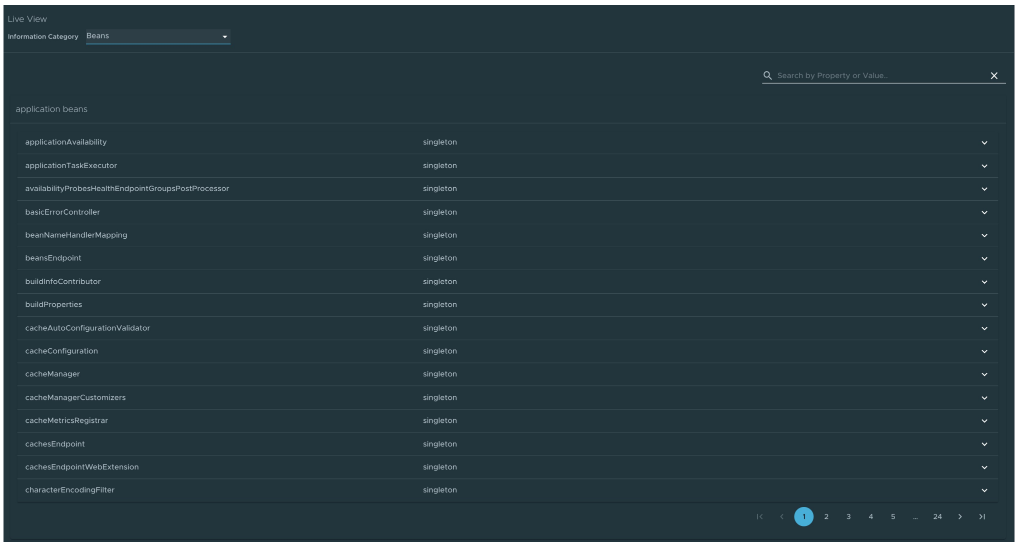
Runnable	Expression	Initial Delay	Interval
com.github.kdvolder.fortune.service.ScheduledTasks.lateTime		1000	50000
com.github.kdvolder.fortune.service.ScheduledTasks.delayedTime		0	15000

fixedRate

Runnable	Expression	Initial Delay	Interval
com.github.kdvolder.fortune.service.FortuneServiceApplication.reportFortne		0	10000
com.github.kdvolder.fortune.service.LogPing.ping		0	2000
com.github.kdvolder.fortune.service.ScheduledTasks.reportCurrentTime		0	5000

Beans Page

To navigate to the Beans page, the user should select the `Beans` option from the `Information Category` dropdown. The beans page provides information about a list of all application beans and its dependencies. It displays the information about the bean type, dependencies and its resource. The user can search by the bean name or its corresponding fields.

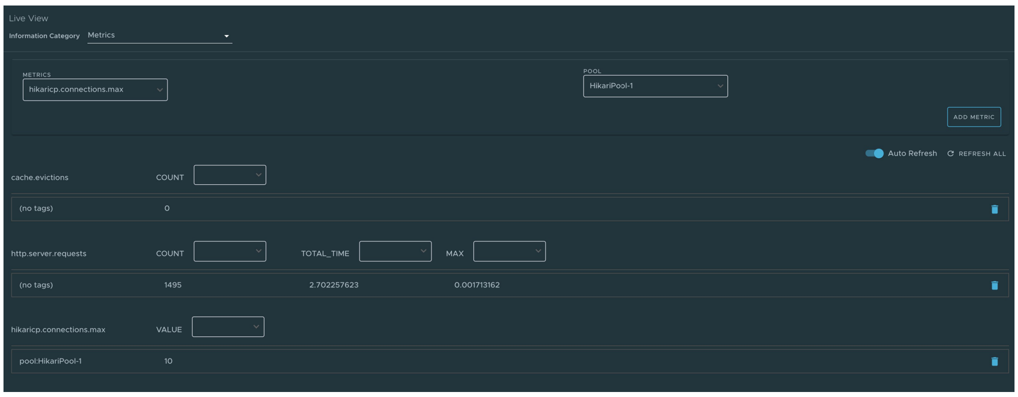


application beans

applicationAvailability	singleton
applicationTaskExecutor	singleton
availabilityProbesHealthEndpointGroupsPostProcessor	singleton
basicErrorController	singleton
beanNameHandlerMapping	singleton
beansEndpoint	singleton
buildInfoContributor	singleton
buildProperties	singleton
cacheAutoConfigurationValidator	singleton
cacheConfiguration	singleton
cacheManager	singleton
cacheManagerCustomizers	singleton
cacheMetricsRegistrar	singleton
cachesEndpoint	singleton
cachesEndpointWebExtension	singleton
characterEncodingFilter	singleton

Metrics Page

To navigate to the Metrics page, the user should select the `Metrics` option from the `Information Category` dropdown. The metrics page provides access to application metrics information. The user can choose from the list of various metrics available for the application such as jvm.memory.used, jvm.memory.max, http.server.request, etc. Once the metric is chosen, the user can view the associated tags. The user can choose the value of each of the tags based on filtering criteria. On clicking *Add Metric* button, the metric is added to the page, which is refreshed every 5 seconds by default. The user can pause the auto refresh feature by deactivating the `Auto Refresh` toggle. The user can also refresh the metrics manually by clicking **Refresh All**. The format of the metric value can be changed according to the user's needs. They can delete a particular metric by clicking on the minus symbol in the same row.



Information Category: Metrics

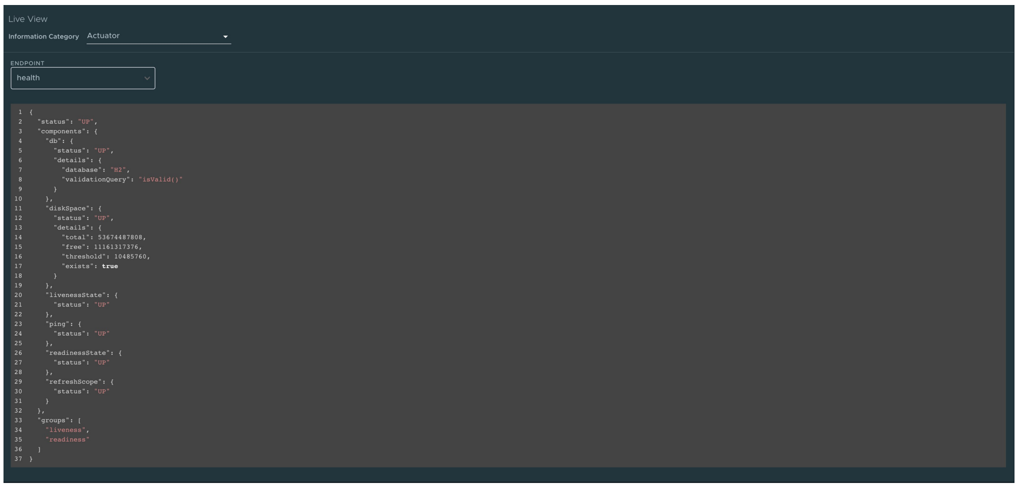
METRICS: hikaricp.connections.max

POOL: HikariPool-1

cache.evictions	COUNT		
(no tags)	0		
http.server.requests	COUNT	TOTAL_TIME	MAX
(no tags)	1495	2.702257623	0.001713162
hikaricp.connections.max	VALUE		
pool:HikariPool-1	10		

Actuator Page

To navigate to the Actuator page, the user should select the `Actuator` option from the `Information Category` dropdown. The actuator page provides a tree view of the actuator data. The user can choose from a list of actuator endpoints and parse through the raw actuator data.



```
{
  "status": "UP",
  "components": {
    "db": {
      "status": "UP",
      "details": {
        "database": "H2",
        "validationQuery": "isValid()"
      }
    },
    "diskSpace": {
      "status": "UP",
      "details": {
        "total": 53674487808,
        "free": 11161317376,
        "threshold": 10485760,
        "exists": true
      }
    },
    "livenessState": {
      "status": "UP"
    },
    "ping": {
      "status": "UP"
    },
    "readinessState": {
      "status": "UP"
    },
    "refreshScope": {
      "status": "UP"
    }
  },
  "groups": [
    "liveness",
    "readiness"
  ]
}
```

Supporting Polyglot Applications

Application Live View supports Spring Boot and Steeltoe applications at the moment.

Support for more application types is planned.

Troubleshooting

My app does not show up in Application Live View UI. Why?

The connector component is responsible for discovering the application and registering it with Application Live View. 1. The app in question needs to be a Spring Boot Application.

1. Make sure an instance of a connector is residing in the namespace of your application.

   ```
   kubectl get pods -n <your namespace> | grep connector
   ```

2. Make sure the actuator endpoints are enabled for your application: `management.endpoints.web.exposure.include: "*"`
3. Make sure the below labels are included in your app deployment yaml

   ```
   tanzu.app.live.view="true"
   tanzu.app.live.view.application.name="<app-name>"
   ```

4. Make sure the convention service workload yaml does not contain property `management.endpoints.web.exposure.include` overrides

My app shows up in Application Live View UI, but the 'Health' page does not show details of health. Why?

The information exposed by the health endpoint depends on the management.endpoint.health.show-details property and this should be set to 'always' as below:

```
management.endpoint.health.show-details: "always"
```

My app is not visible in Application Live View UI, even though the actuator endpoints are enabled. Why?

1. Check the port on which actuator endpoints are configured. By default, they are enabled on the application port. If they are configured on a port different from the application port, make sure to set the labels in your application deployment yaml:

   ```
   tanzu.app.live.view.application.port: "<application port>"
   tanzu.app.live.view.application.actuator.port: "<actuator port>"
   ```

2. Check the path on which the application and actuator endpoints are configured. If they are configured on a different paths, make sure to set the labels in your application deployment yaml:

   ```
   tanzu.app.live.view.application.path: "<application path>"
   tanzu.app.live.view.application.actuator.path: "<actuator path>"
   ```

How do I verify whether the labels in my workload yaml are working fine?

1. Verify the application live view convention webhook is running properly using `kubectl get pods -n app-live-view | grep webhook`
2. Verify the conventions controller is running properly using `kubectl get pods -n conventions-system`
3. Run `kubectl get podintents.conventions.apps.tanzu.vmware.com <your workload name> -oyaml` to check if the conventions are applied properly to the PodSpec. If everything works correctly, the status will contain a transformed template that includes the labels added as part of your workload yaml.

   For example:

```
status:
conditions:
- lastTransitionTime: "2021-10-26T11:26:35Z"
  status: "True"
  type: ConventionsApplied
- lastTransitionTime: "2021-10-26T11:26:35Z"
  status: "True"
  type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      conventions.apps.tanzu.vmware.com/applied-conventions: |-
        appliveview-sample/app-live-view-connector
        appliveview-sample/app-live-view-appflavours
        appliveview-sample/app-live-view-systemproperties
    labels:
      tanzu.app.live.view: "true"
      tanzu.app.live.view.application.flavours: spring-boot
      tanzu.app.live.view.application.name: petclinic
  spec:
    containers:
    - env:
      - name: JAVA_TOOL_OPTIONS
        value: -Dmanagement.endpoint.health.show-details=always -Dmanagement.en
dpoints.web.exposure.include=*
    image: index.docker.io/kdvolder/alv-spring-petclinic:latest@sha256:1aa7bd22
8137471ea38ce36cbf5ffcd629eabeb8ce047f5533b7b9176ff51f98
    name: workload
    resources: {}
```

I see stale information in App Live View, i.e. I find my app in the UI but it is actually an old instance that no longer exists while the new instance doesn't show up yet

Check the App Live View connector pod logs to see if the connector is sending updates to the backend. Also, as a workaround, try deleting the connector pod so it gets recreated.

```
kubectl -n app-live-view delete pods -l=name=application-live-view-connector
```

In TAP GUI, I see `No live information for pod with id` error sometimes

This could happen because of stale information in App Live View, i.e. it is actually an old instance that no longer exists while the new instance doesn't show up yet. The workaround is to delete the connector pod so it gets recreated.

```
kubectl -n app-live-view delete pods -l=name=application-live-view-connector
```

Can I override the labels set by the Application Live View Convention Service for the Workload deployment in TAP?

No, the labels `tanzu.app.live.view`, `tanzu.app.live.view.application.flavours` and `tanzu.app.live.view.application.name` cannot be overriden and the defaults set by the Application Live View Convention Server would be used. However, if the users intend to override management.endpoints.web.exposure.include or management.endpoint.health.show-details, the environment properties can be overriden in application.properties or application.yml in the Spring Boot Application before deploying the Workload in TAP. The environment properties updated by the users in their application would take precedence over the defaults set by Application Live View Convention Server.

My app has management.endpoints.web.base-path and management.server.port set. How to configure the actuator labels?

If the custom actuator context path is configured like below (for example):

```
management.endpoints.web.base-path=/manage
management.server.port=8085
```

The connector can be configured like below:

```
tanzu.app.live.view.application.actuator.path=/manage   (manage is the custom actuator
 path set on the application)
tanzu.app.live.view.application.actuator.port=8085   (8085 is the custom management se
rver port set on the application)
```

The sidecar can be configured like below:

```
app.live.view.sidecar.application-actuator-path=/manage  (manage is the custom actuato
r path set on the application)
app.live.view.sidecar.application-actuator-port=8085  (8085 is the custom management s
erver port set on the application)
```

My app has actuator endpoints exposed at root (/) and the UI does not show any information

Application Live View will not be able to display the application details when the application is exposing the actuator endpoint on root (/) . This is due to conflict in the actuator context path and application default context path.

I'm not able to override the actuator path in the labels as part of the Workload deployment. What do I do?

The changes to add/override labels/annotations in the `Workload` are in progress. The changes from the `Workload` need to be propagated up through the supply chain in order for the PodIntent to see the new changes.

Release Notes

This topic contains release notes for Application Live View for VMware Tanzu Release v1.0.

v1.0.3 release

Release Date: March 31, 2022

Features

New features and changes in this release:

Components

- Updated Spring Framework to `5.3.18` to address `CVE-2022-22965`
- Updated Jackson-BOM to `2.13.2.2022032` to address `CVE-2020-36518`

v1.0.2 release

Release Date: February 8, 2022

Features

New features and changes in this release:

Components

- Includes changes to the App Live View connector to handle stream reset exceptions
- Updated pod security policies for App Live View Components
- Increased requests and limits for App Live View Connector to fix pod restarts
- Updated Spring Boot 2.5.7 to 2.5.8
- CVE vulnerability fix - to update protobuf-java to 3.19.2 in connector

Known issues

None

v1.0.1 release

Release Date: December 17, 2021

Features

New features and changes in this release:

Components

- Includes changes to the App Live View Convention Service to be compatible with change in sBOM location and format
- Updated the log4j versions for the App Live View components to 2.16.0

Known issues

None

v1.0.0 beta release

Release Date: November 29, 2021

Features

New features and changes in this release:

Packages

Application Live View installs two TAP packages for full and dev profiles in this release. Application Live View Package (run.appliveview.tanzu.vmware.com) contains Application Live View Backend and Connector components. Application Live View Conventions Package (build.appliveview.tanzu.vmware.com) contains Application Live View Convention Service only.

Components

- Includes changes to the App Live View connector to fall back to annotations if labels are not present or set to invalid value
- Removed `server.service_type` in values.yml
- Removed CRD controller
- Removed UI : The UI has been removed and rewritten to be part of the TAP GUI that is provided with the Tanzu Application Platform.

Known issues

None

v0.3.0 beta release

Release Date: October 25, 2021

Features

New features and changes in this release:

Components

- Created namespace `app-live-view` as part of package installation. Removed `server_namespace` in values.yaml.
- Changed the `service_type` in values.yaml to Cluster IP as default value (instead of LoadBalancer)
- Includes the native version of App Live View Connector
- App Live View backend service is exposed on port 80 so that the user can directly enter the service URL without specifying the port.

Known issues

None

v0.2.0 beta release

Release Date: September 27, 2021

Features

New features and changes in this release:

Components

- Created namespace `app-live-view` as part of package installation. Removed `server_namespace` in values.yaml.
- Changed the `service_type` in values.yaml to Cluster IP as default value (instead of LoadBalancer)

- Includes the native version of App Live View Connector
- App Live View backend service is exposed on port 80 so that the user can directly enter the service URL without specifying the port.
- Certificate `appliveview-webhook-cert` is configured to be in app-live-view namespace so the PodIntent runs without errors for a Workload.

Known issues

None

v0.1.0 beta release

Release Date: September 1, 2021

Initial release

Known issues

The certificate ‘appliveview-webhook-cert’ is found missing in app-live-view namespace and hence the PodIntent fails on the workload as below:

```
Message:               failed to authenticate: unable to find valid certificaterequest
s for certificate "app-live-view/appliveview-webhook-cert"
Reason:                CABundleResolutionFailed
Status:                False
Type:                  ConventionsApplied
```