

Kohl's Bigdata Platform

DevOps User Manual

For

BigData Microservices

Version 0.1

**Kohl’s**

Prepared by

BigData DevOps Team

Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Author** | **Description** | **Preparation Date** | **Reviewed By** | **Effective Date** |
| 0.1 | Smitha M | Technical manual intended to assist DevOps L1/L2 and L3/L4 teams |  |  |  |
| 0.2 | Smitha M | Technical manual intended to assist DevOps L1/L2 and L3/L4 teams |  |  |  |
|  |  |  |  |  |  |

Approvals

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Approval Date** | **Approver Name** | **Comments** |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction: 5](#_Toc514837089)

[1.1 Overview : 5](#_Toc514837090)

[1.2 Purpose: 5](#_Toc514837092)

[1.3 Definitions, Acronyms, and Abbreviations 5](#_Toc514837093)

[1.4 References 5](#_Toc514837094)

[2.Microservices 6](#_Toc514837095)

[2.1 Introduction: 6](#_Toc514837096)

[3. BigData Architecture for Kohls: 6](#_Toc514837097)

3.1 Component Overview of BigData Architecture 8

3.2 Workflow Explanation of Architecture Diagram 9

[4. Deployment of Microservices of Bigdata: 12](#_Toc514837098)

[4.1 CI workflow diagram 13](#_Toc514837099)

[4.1.1 Gerrit repo: 14](#_Toc514837101)

[4.1.1.1 Introduction: 14](#_Toc514837102)

[4.1.1.3 To add roles to existing Gerrit Repo : 16](#_Toc514837103)

[4.1.2 JENKINS CI/CD PIPELINE CREATION 16](#_Toc514837104)

[4.1.2.1 Step by Step bootstrap example 17](#_Toc514837105)

[4.2 ENVIRONMENT CREATION 19](#_Toc514837106)

[4.2.1 Prerequisites: 20](#_Toc514837107)

[4.2.2 Operator Workstation: 20](#_Toc514837108)

4.2.3 Component way of Creating an Environment 15

4.2.3.1 Runner Creation 16

4.2.3.2 Modying Code-base 16

4.2.3.3.Bootstrapping Environment 16

4.2.3.4 Environment Creation 17

4.3 Deployment Steps using Tonomi 31

5 Appendix 44

5.1 Gerrit Repo Cloning URL(s) 44

5.2 DNS for Bigdata Components 44

5.3 Akamai and ELB URLs 45

5.4 Bigdata Microservices Component Details on Tonomi 46

5.5 GCP URL 46

5.6 Contact Detail 46

**1. Introduction**

## Overview:

## The idea behind microservices architecture is to build your application as, many independent services rather than one large code base. Rather than accessing most of your data using large databases, communication is often handled with API calls between the services, with each service having its own lightweight database. Kohls has 8 different Bigdata microservices in place, they are as follows:

1. BPBS (Business portal Backend Service)
2. PAP (Personalized Assets processing)
3. EDE (Experienced Design Engine)
4. EDE Prev (Experienced Design Engine preview)
5. INNOAPP
6. WORKBENCH
7. MFP (Meta feed Processing Service)
8. KIRAP

## Purpose:

This technical guide provides information for two types of audiences:

1. L3/L4 DevOps engineers who are responsible for Deploying and supporting Bigdata Microservices.

## Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Abbreviations** | **Details** |
|  |  |
|  |  |
|  |  |
|  |  |

## References

|  |  |
| --- | --- |
| **Document Name** | **Details** |
|  |  |

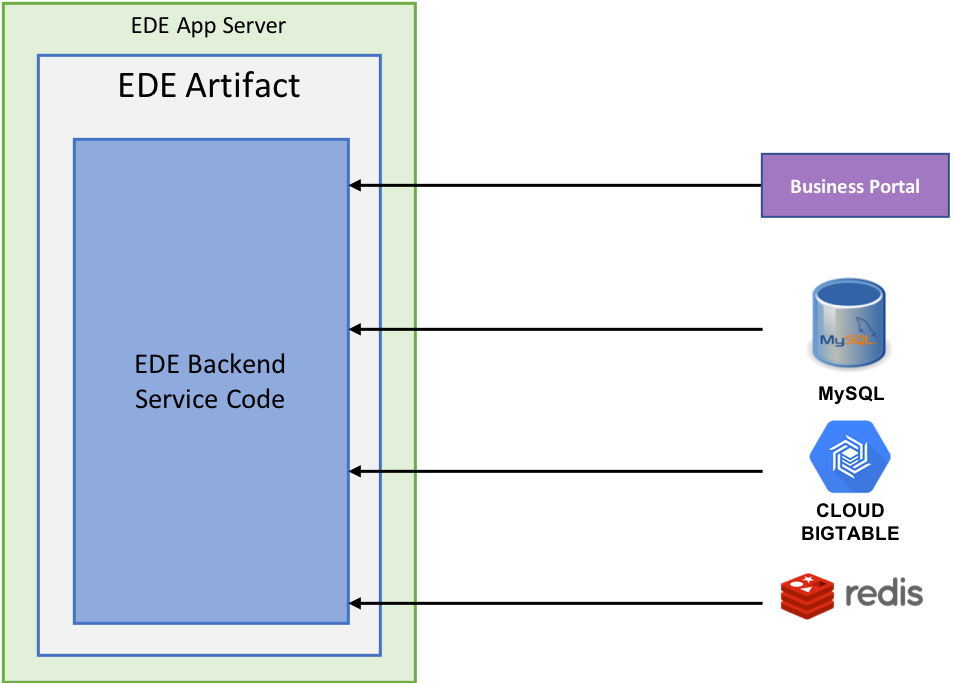
# 2. Microservices

# 2.1 Introduction:

Microservice architecture is a method of developing software applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal. We are using the below microservices in Kohls, which are described as follows:

**Experience Decision Engine (EDE)**

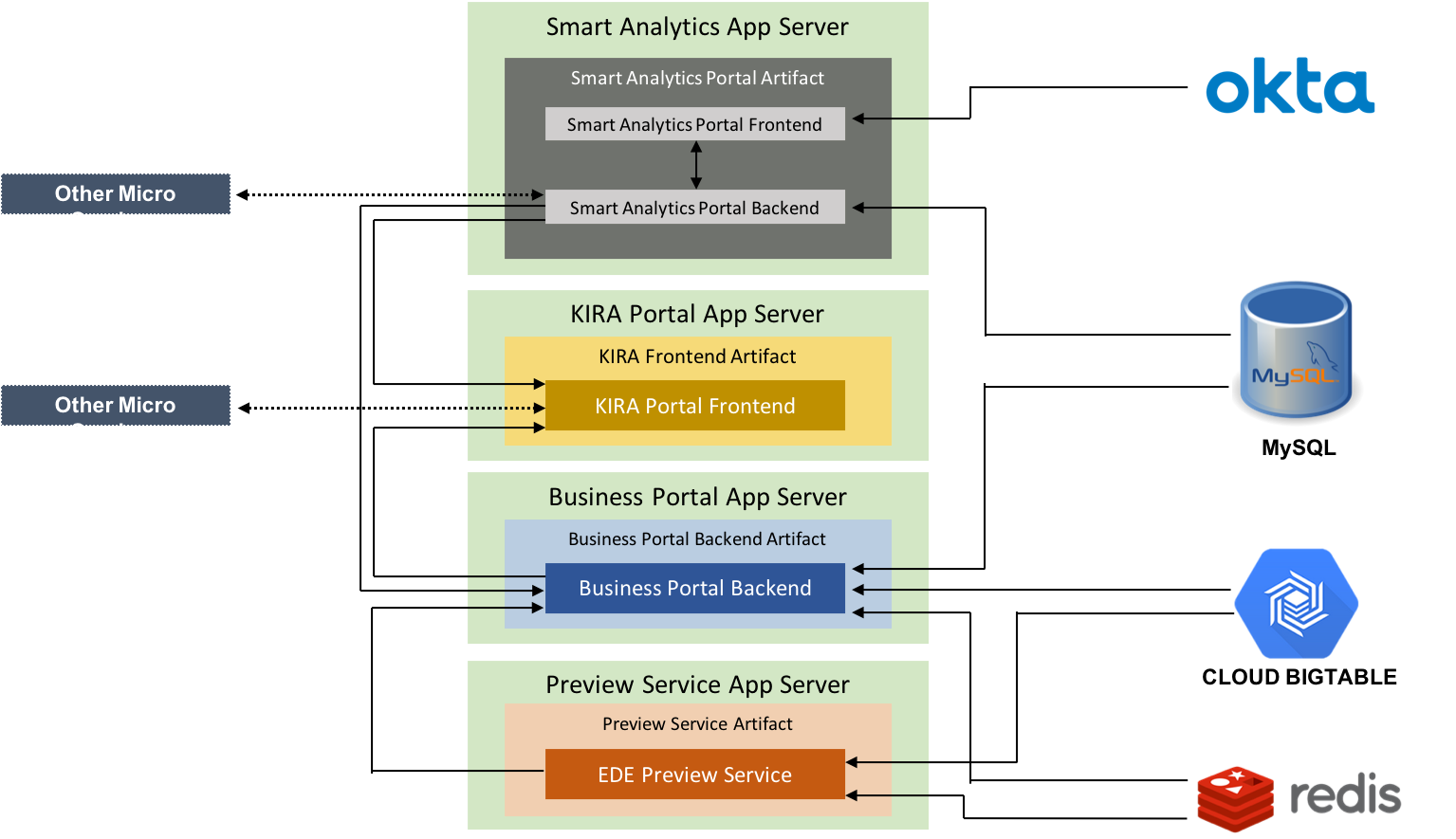
1. Kohl’s Experience Decision Engine (**EDE**) in Google Cloud Platform is the middle-tier service which is the equivalent version of the EDE on Prem version, which was designed to replace the legacy Recommendation Engine Client JAR’s functionality in accessing the Kohl’s Bigdata Platform Recommendations System.
2. EDE itself will be a hosted middle-tier service which will provide access to Bigdata recommendations through a RESTful interface. With the deployment of EDE, channels will no longer be required to go through Kohl’s Open API to retrieve Bigdata recommendations. Following diagram illustrates the overall functionality of EDE.

****

**Fig1 Experience design Engine GCP architecture**

**Business Portal (BP)**

1. Business Portal Artifact itself contains two separate applications that are essentially supposed to do two things, which are as follows:
2. Smart Analytics Portal is supposed to provide a single portal to access all Big Data tools.
3. Business Portal (aka KIRA Portal) is supposed to be used by Business team to configure KIRA business rules.



**Fig2 Business Portal Engine GCP architecture**

**PAP**

# Big Data Personalized Assets will enable the personalization of assets shown to customers in various pages of the Kohl’s customer facing channels (E.g. WebStore, TCom, MCom).

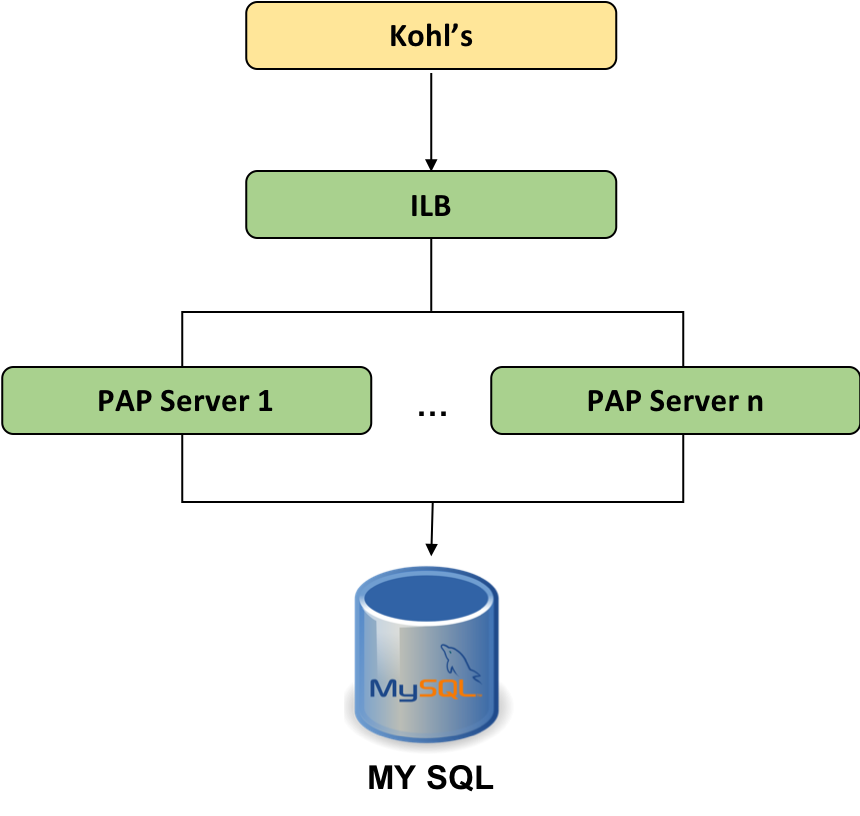
# Without personalized assets, all customers visiting the site will see the same static assets

configured for each page. With the personalized assets, Big Data will use the

customer’s propensity scores to re-rank multiple assets that are configured for the

same placement of a given channel page, to identify the most suitable ranked order

of the assets to display based on that particular customer’s preferences.



**Fig3 Personalized Assets Processor Engine GCP architecture**

**Meta Feed Processor Engine**

# This will be a new micro-service which will read the feeds from Cloud Storage, process those feed files, and update the metadata caches in the central Redis cluster which will be a shared centralized cache between all micro-services (E.g. All EDE servers, All BP servers).

# Advantages:

# Consistency of metadata across micro-services.

# Simplifies the micro-services.

# Enables micro-service auto scaling.

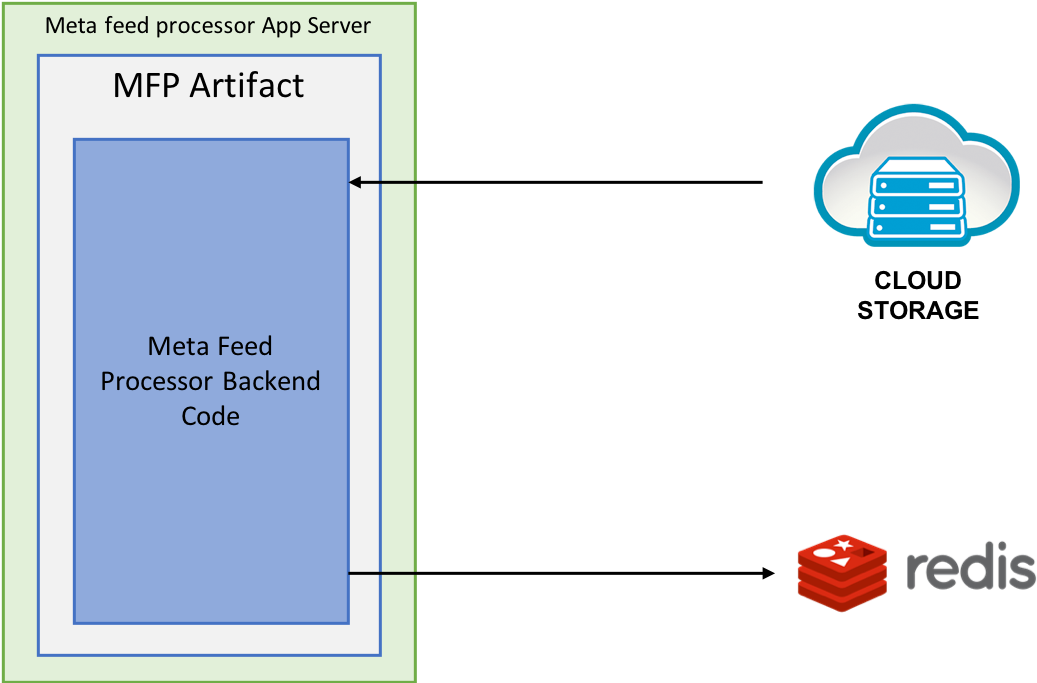
# Disadvantages:

# Cost of Redis.

# Single Point of Failure.

# SPOF of Redis can be catered with Redis Sentinal & Redis Cluster.

# SPOF of Meta Feed Processor Engine can be prevented from an active-passive deployment.



**Fig4 Meta Feed Processor Engine GCP architecture**

**WorkBench (WB)**

1. Provides Operational Insights at application level
2. Tool to Validate, Troubleshoot and Isolate issues

**Innovation Application**

1. This application is used to build a UI that simulates a "demo channel" to showcase EDE and BigData recommendation capabilities.
2. UI layout will consist of static and dynamic slots that can configured through the portal

# 3. Bigdata Architecture for Kohls:

**3.1 Component Overview of Bigdata Architecture:**

**Okta:** The Okta Authentication API provides operations to authenticate users, perform

multi-factor enrollment, verification, recover forgotten passwords, and unlock

accounts Primary authentication allows you to verify username and password credentials

for a user.

**Redis Cache**: Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyper logs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

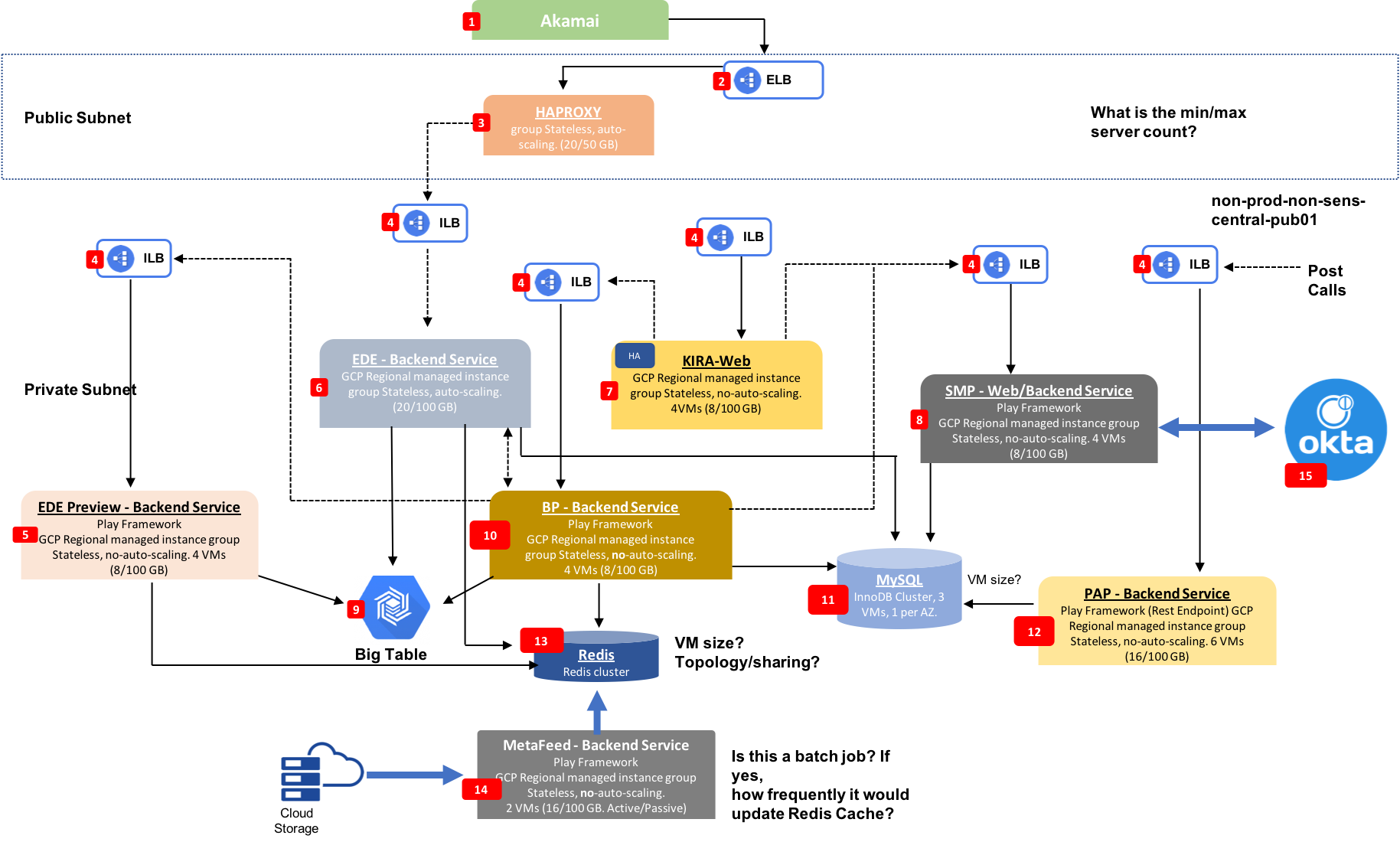
**Bigtable**: is designed to handle massive workloads at consistent low latency and high throughput, so it's a great choice for both operational and analytical applications, including IoT, user analytics, and financial data analysis.

**MySQL:** MySQL is being used as relational Database systems used for storing and analyzing the huge amount of data that is being stored. MySQL is good at handling highly concurrent accesses to transactional data on a single machine.

**External Load Balancer:** We use External Load Balancer (ELB) which is public facing for EDE application.

**HAProxy:** stands for High Availability Proxy, is a popular open source software TCP/HTTP Load Balancer and proxying solution which can be run on Linux, Solaris, and FreeBSD. Its most common use is to improve the performance and reliability of a server environment by distributing the workload across multiple servers (e.g. web, application, database). It is used in many high-profile environments, including: GitHub. HAProxy is being used for SSL termination.

**AKAMAI:** Acts like a CDN, when a Request hits AKAMAI, it will be served from the cached edge location.



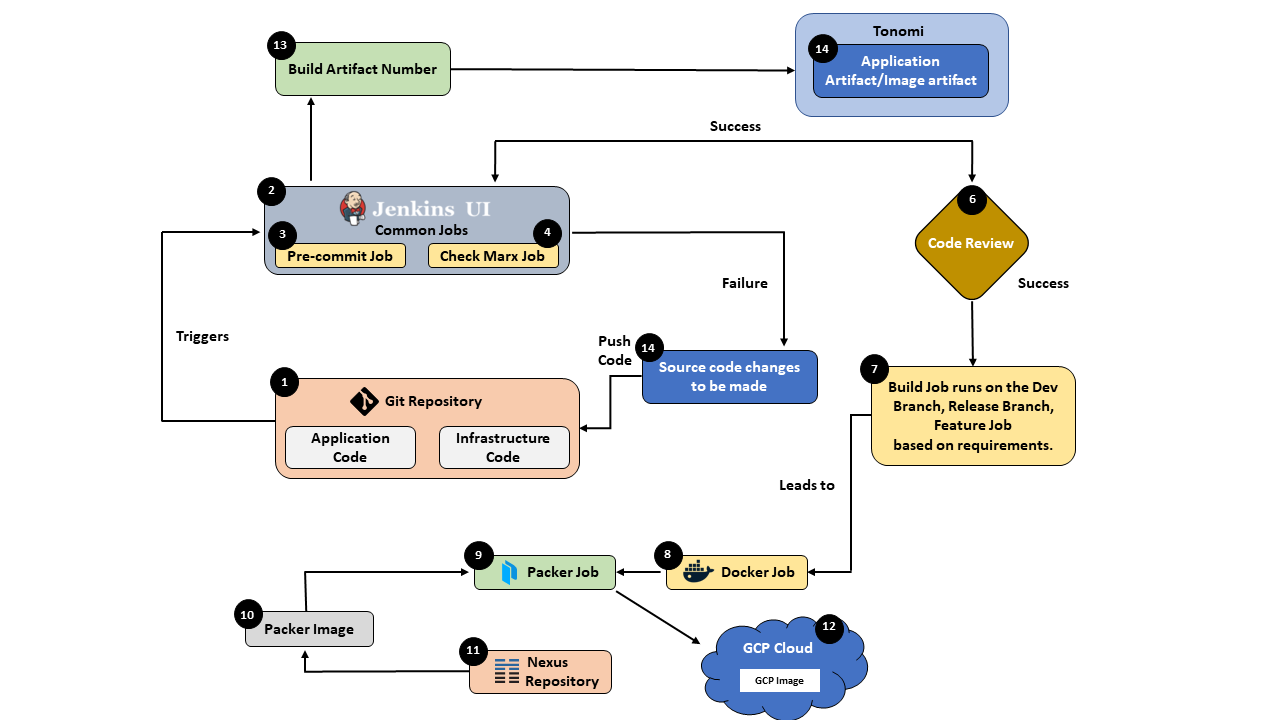
# 4. Deployment of Microservices of Bigdata:

Microservice architecture is to buildyour application as many independent servicesrather than one large code base. There are eight different Microservices on Bigdata, all of which are deployed following the same procedure. Application team and the infrastructure team have two different repos on Gerrit, to which they push their codes. All the changes that are made to the code with respect to infrastructure or the application functionalities will trigger a pipeline of Jenkins jobs. After successful execution of Jenkins pipeline jobs, a build number will be generated which will later be fed to the Tonomi to successfully deploy the concerned artefact

We follow the below steps to deploy these Microservices.

1. CI Workflow Diagram
   1. Gerrit Repo Creation
   2. Jenkins CI/CD Pipeline Creation
2. Environment Creation
   1. Bootstrapping Environment Creation
   2. Environment creation from Bootstrapped Environment
3. Application Deployment Using Tonomi

**4.1 CI work flow diagram:**



# 

# Fig1 CI Work flow Diagram

# 

# Description of the CI diagram:

1. Firstly, A Gerrit Repo is created which includes both the application code and the infrastructure code.
2. The Code which is newly added to this repo, or the changes made to the code by the application team as well as the infrastructure team are pushed to this Repository.
3. These new pushes will trigger the jobs on Jenkins.
4. Jenkins pipeline mainly includes the following Jobs:
   1. Pre-commit
   2. Checkmarkx
   3. Build
   4. Docker
   5. Packer

1. The Pre-commit and Checkmarx jobs are common for all the deployments, hence they run on the Common Branch.
2. Build, Docker and Packer Jobs run on Dev, Feature or Release branches based on the requirements.
3. When the first two jobs, Pre-commit and Checkmarkx have succeeded, the new code/changed will have to be reviewed.
4. If any of these two jobs fail, the concerned teams will be notified to make changes on the code and push it to Gerrit Repo.
5. The code is reviewed and approved, it will be obvious to run the other three jobs on any one of the following branches, namely:
   1. Dev
   2. Feature
   3. Release.
6. Bild job will give the Build Artifact, Infrastructure code artifact and Terraform artifact, which will be pushed to Nexus Repository.
7. Build Job will be succeeded by the Docker Job which will later be succeeded by Packer Job.
8. Packer Job will pull the packer code from the Nexus Repository and run the job.
9. After the Packer Job has run successfully to give a GCP image which is later stored on the GCP cloud.
10. Build Artifact Number will be generated by the Jenkins, after the success of the Jenkins which is manually copied and fed to the Tonomi Tool which will deploy the required artifacts for us.

# 

# 4.1.1 Gerrit repo:

# 4.1.1.1 Introduction:

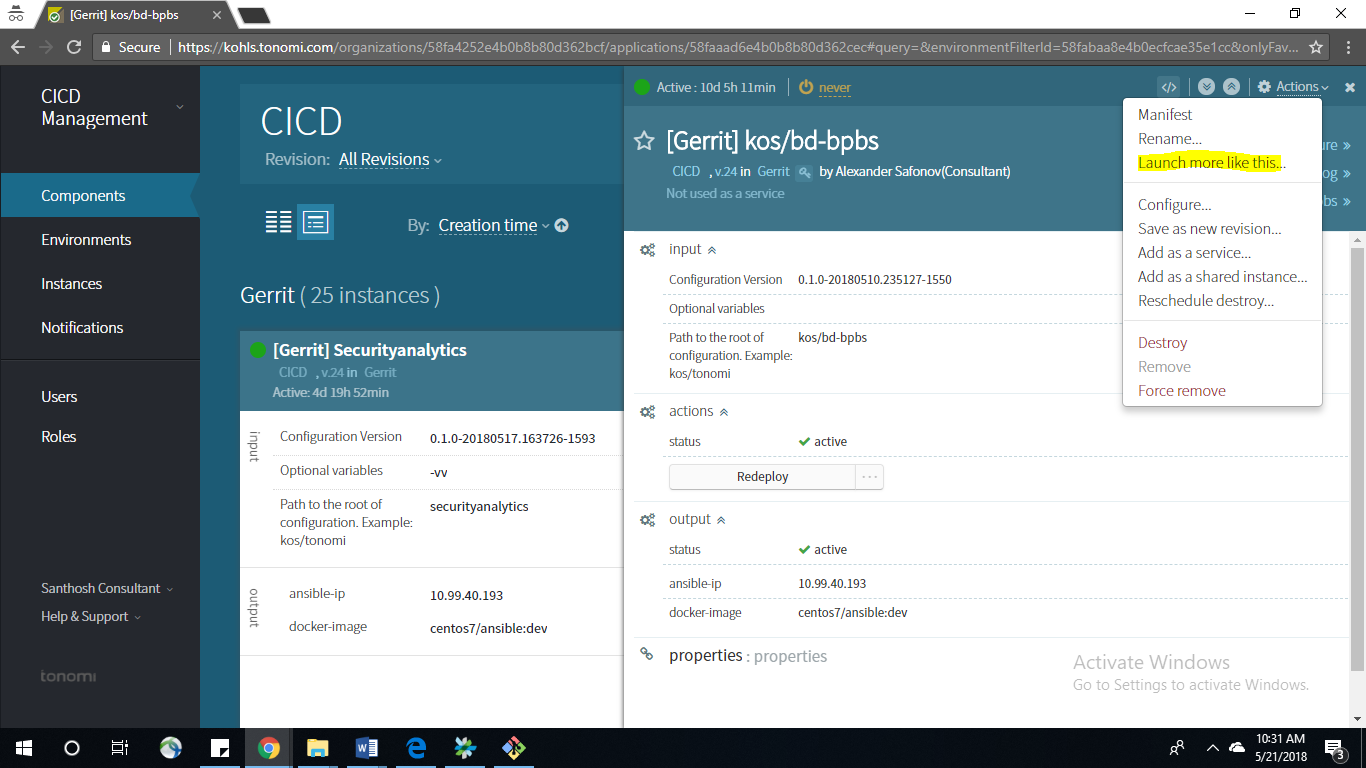
Gerrit is firstly a staging area where changes can be checked over before becoming a part of the code base. It is also an enabler for this review process, capturing notes and comments about the changes to enable discussion of the change. This is particularly useful with distributed teams where this conversation can’t happen face to face

Gerrit is intended to provide a lightweight framework for reviewing every commit before it is accepted into the code base. Changes are uploaded to Gerrit but don’t become a part of the project until they’ve been reviewed and accepted.

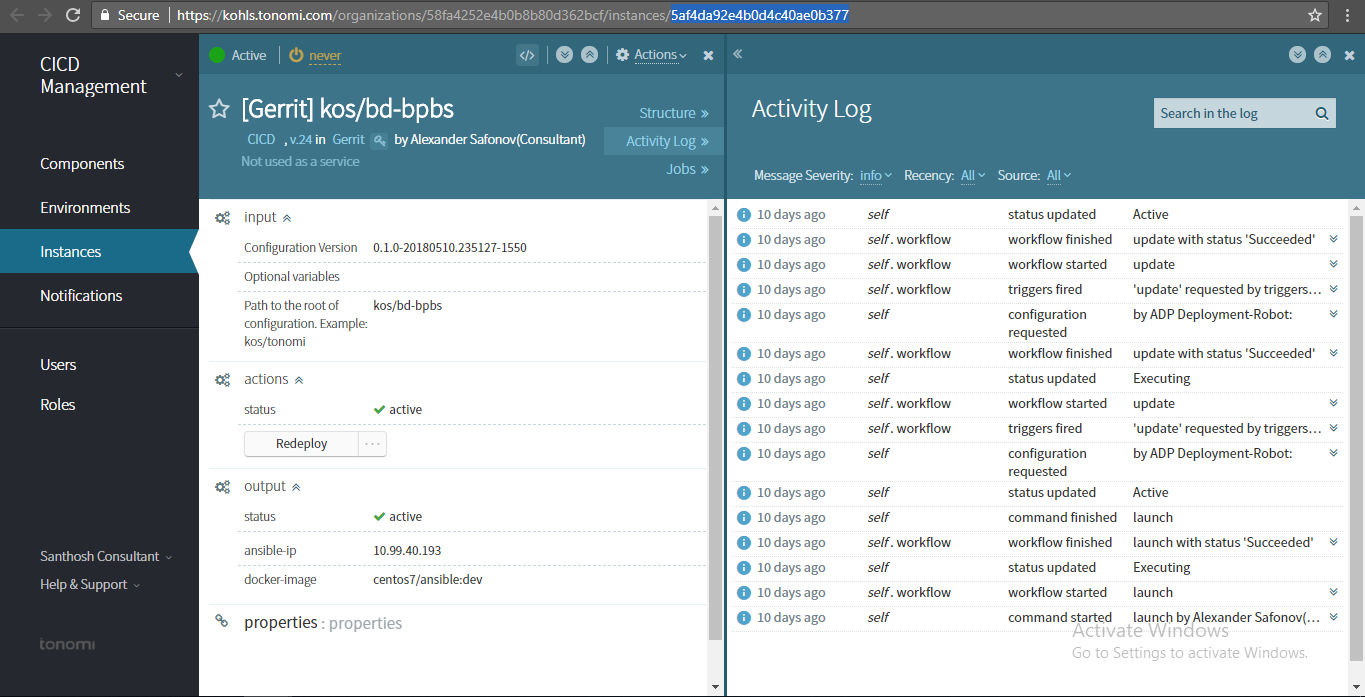
# 

#### 4.1.1.2 Gerrit Repo Creation

1. Find project directory in [cicd-ansible](https://cicd-gerrit.kohls.com:8443/gerrit/#/admin/projects/cicd-ansible) repo in “vars/gerrit” folder
2. Create the required directory for example: kos/bd-sample
3. Copy the main.yml and instances file from the existing directory
4. Replace the Project name with the project name as kos/bd-sample:
5. Now go to Tonomi under CICD organization:



1. Under Gerrit Environment launch CICD component to create the instance and copy the instance id:



1. Now go back to kos/bd-sample directory (instances file) and put the instance id
2. Now push the code in Gerrit
3. This triggers pre-commit job followed by review and dev job.

# 

# 4.1.1.3 To add roles to existing Gerrit Repo:

1. Find project directory in [cicd-ansible](https://cicd-gerrit.kohls.com:8443/gerrit/#/admin/projects/cicd-ansible) repo in “vars/gerrit” folder
2. Add user’s email address to the required groups (engineers, leads, release-approvers) in the “main.yml” file

Example: There is a request to add permissions for User1to participate in kos/cnc-services development.

* + - 1. Open the vars/gerrit/kos/cnc/main.yml file
      2. Find “kos/cnc-services” > “groups” > “engineers” and add “User1@kohls.com” to the members list.

# 4.1.2 JENKINS CI/CD PIPELINE CREATION

Common Pipeline Jobs for both Middleware and Application components:

|  |  |  |
| --- | --- | --- |
| **Branches** | **Jobs** | **Remarks** |
| Common | kOS-<PROJECT>-Precommit | Common triggered by new change set in gerrit  (Verify +1/-1 mark) |
| Common | kOS-<PROJECT>-Checkmarx-Verification | Triggered by new change set in gerrit  (Verify +1/-1 mark) |
| Dev | kOS-<PROJECT>-Build-Dev | Triggered by new commit in dev branch |
|  | kOS-<PROJECT>- Build-Docker-Dev | Common triggered by kOS-<PROJECT>-Build-Dev |
|  | kOS-<PROJECT>- Build-Packer-Dev | Triggered by kOS-<PROJECT>-Build-Docker-Dev |
| Feature | kOS-<PROJECT>-Build-Feature | Triggered by new commit in dev branch |
|  | kOS-<PROJECT>- Build-Docker-Feature | Common triggered by kOS-<PROJECT>-Build-Feature |
|  | kOS-<PROJECT>- Build-Packer-Feature | Triggered by kOS-<PROJECT>-Build-docker-Feature |
| Release | kOS-<PROJECT>-Build-Release | Triggered by new commit in dev branch |
|  | kOS-<PROJECT>- Build-Release | Common triggered by kOS-<PROJECT>-Build-Docker-Release |
|  | kOS-<PROJECT>- Build-Release | Triggered by kOS-<PROJECT>-Build-Ps |

# 4.1.2.1 Step by Step bootstrap example

Add a fully operational Jenkins pipeline for new bigdata project.

**Steps:**

1. Create a new folder “vars/jenkins/kos/<project\_short\_name>” under CICD-ANSIBLE Repo.

2. Create two files named “instances” and “main.yml” in the folder.

3. Fill the “main.yml” file with the default pipeline description

***other\_variables:***

***jenkins\_project\_main\_view:***

***<PROJECT\_VIEW>***

***​jenkins\_main\_email\_list: ​"<EMAILS>"***

***projects\_env:***

***<PROJECT>:***

***repository: ​<GERRIT\_PROJECT>***

***project\_pipeline\_jobs: <NON\_DEFAULT\_PIPELINE\_JOB\_1> <NON\_DEFAULT\_PIPELINE\_JOB\_2***

**where:**

|  |  |
| --- | --- |
| EMAILS | Comma separated list of email addresses to |
|  | send notifications to |
|  |  |
| PROJECT | The name of the project that will be used for |
|  | naming the Jenkins jobs |
|  | (Example: “ACC-Services”) |
|  |  |
| GERRIT\_PROJECT | The name of the repository in Gerrit |
|  | (Example: “kos/acc-services”) |
|  |  |
| NON\_DEFAULT\_PIPELINE\_JOB\_N | Declaration of an additional behavior in the |
|  | pipeline that adds an extra jobs configuration |
|  | or disables some steps of the default pipeline. |
|  |  |

1. To apply the changes and create Jenkins jobs there are two available options.
2. Automated application of changes

a. Launch new instance of ​ [CICD component in Tonom](https://kohls.tonomi.com/organizations/58fa4252e4b0b8b80d362bcf/applications/58faaad6e4b0b8b80d362cec)i ​if it was not created before.

b. Launch an Instance in Jenkins with the below steps: Using the below

configuration details:

***Environment:Jenkins***

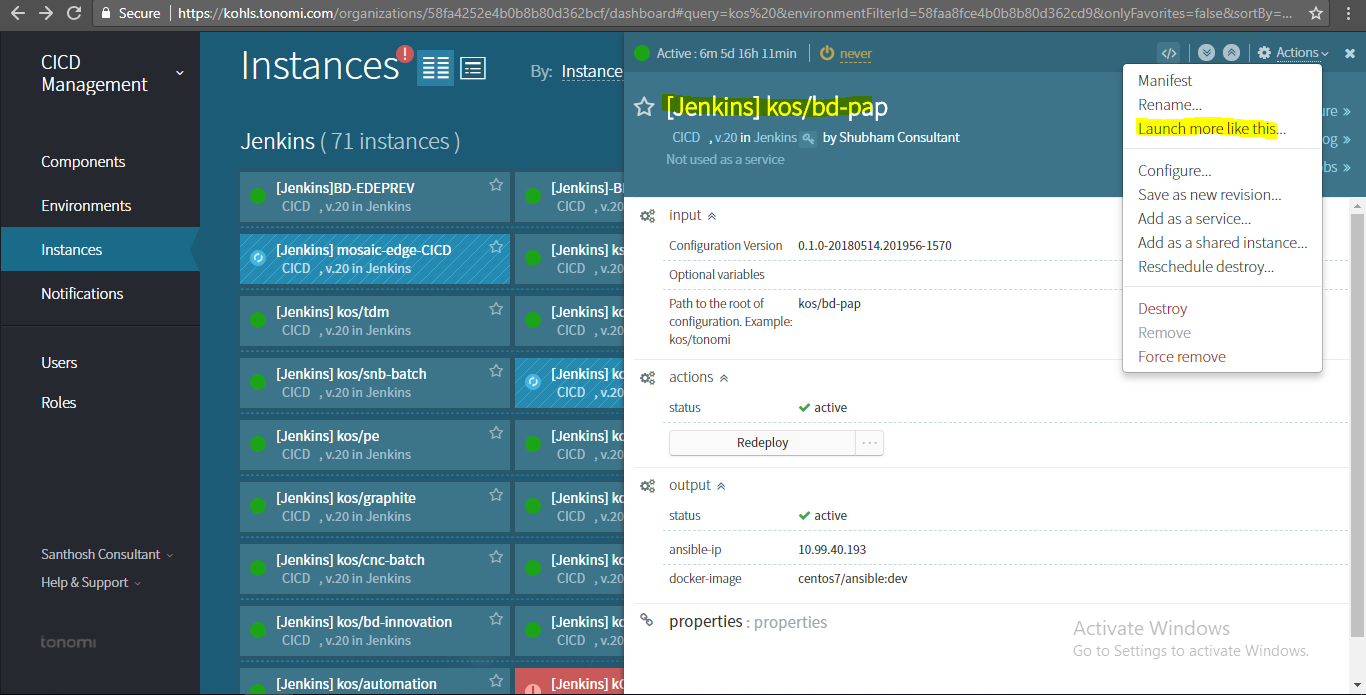
***for: ​unlimited time***

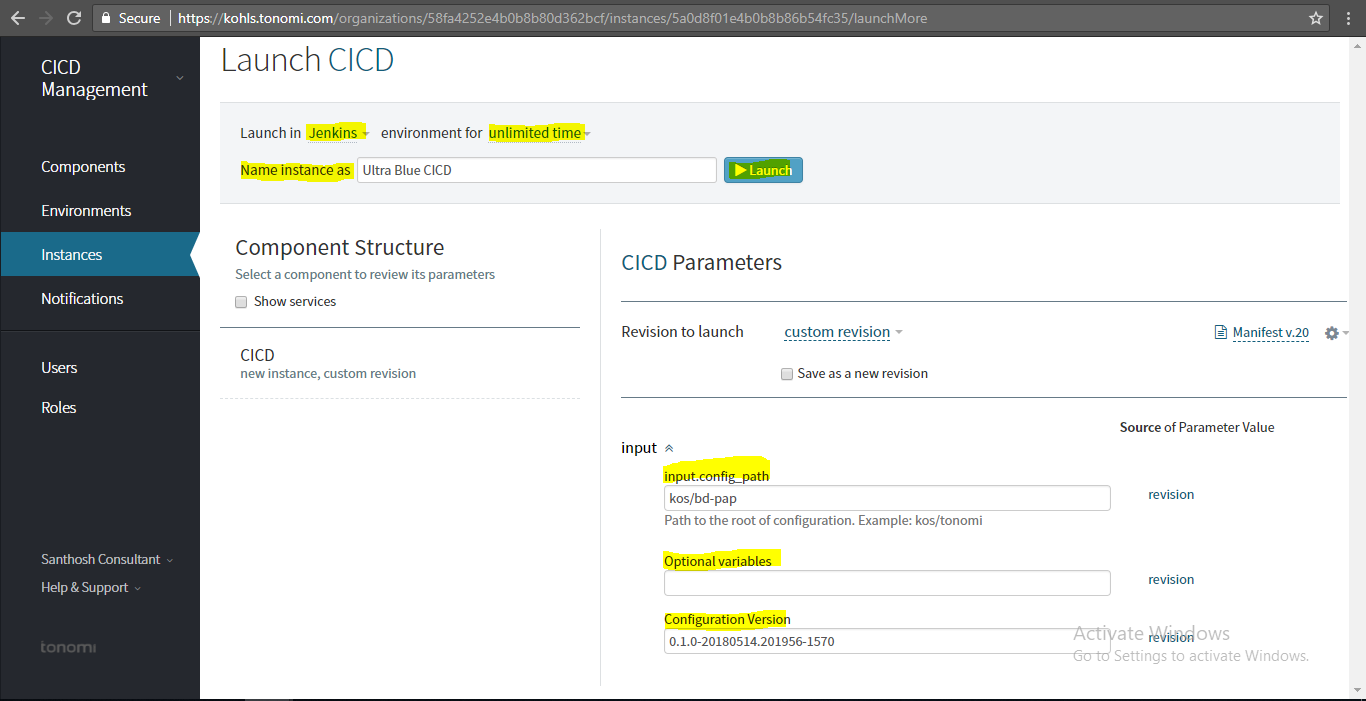
***Name instance as: ​[Jenkins] BigData <PROJECT>***

***input.config\_path: ​kos/<PROJECT>​(the folder path relative to the “vars/jenkins” folder)***

***Optional variables: leave empty***

***Configuration Versions: artifact version can be retrieved from the***  [***CICD-Cicd-Ansible-Build***](https://cicd-jenkins.kohls.com:8443/jenkins/job/CICD-Cicd-Ansible-Build-Dev/)***-Dev​ job:***





6. Retrieve the ID of the created Tonomi’s instance and put it to the “instances” file

# 

# 

# 4.2 ENVIRONMENT CREATION

# 4.2.1 Prerequisites:

1. Short (technical) name of the environment (matching /[a-z][a-z0-9]+/).
2. GCP parameters of the environment: project name, subnetworks.
3. List of GCP service accounts for application
4. The operator is allowed to do one of the followings:
5. GCP permissions to perform Terraform deployments

Or

Permitted to create and get credential of "terraform-provisioner" GCP service account.

Or

The credentials of JSON file needs to be provided for the operator.

# 4.2.2 Operator Workstation:

1. Software installed at the operator’s computer are as below:
   1. Google Cloud SDK.
   2. Hashicorp Terraform 0.9.4 (**exactly this version**).
   3. Hashicorp Consul CLI 0.7.5 (CLI parameters for 0.8.x differ).
   4. Hashicorp Vault CLI 0.7.x.
   5. [jq](https://stedolan.github.io/jq/) version 1.4 or higher.

**Authorize at GCP and configure GCP SDK:**

1. Authorize at GCP project of the environment with  
   **gcloud init**  
    If it’s already done - select the corresponding gcloud configuration with  
    **gcloud config configurations activate <configuration>**
2. [Configure Google Cloud Application Default Credentials](https://developers.google.com/identity/protocols/application-default-credentials#toolcloudsdk) (if is not already done) with  
   **gcloud auth application-default login**
3. Obtain credential of "terraform-provisioner" GCP service account as a JSON file. Then  
    **export GOOGLE\_CREDENTIALS=$(cat <gcp\_credentials\_json>)**
4. TCP ports 8000-8999 at GCE VMs are reachable from the operator computer.
5. If the operator doesn’t use direct access to Kohl’s network, it is necessary to add the following proxy server to the operator’s environment.
6. **# export http\_proxy=proxy.kohls.com:3128**

**4.2.3 Component Way of Creating an Environment:**

Steps for Creating an Environment:

1. Runner Creation
2. Modifying Code-Base
3. Bootstrapping Environment
4. Creating a new environment

**4.2.3.1 RUNNER CREATION:**

Component way of creating env:

1. Runner is the first component that has to be created in order to create a runner.
2. Steps to Create a runner
3. Gerrit repo used to create a runner: cloud-env-config
4. We have to bootstrap the runner instance using the cloud-env-config
5. Steps to Bootstrap:
6. cd cloud-env-config
7. cd environments
8. mkdir testlle
9. Copy the contents of any other folder in the environments into testlle folder
10. We have 3 files
    * + backend.tf, terraform. tfvars, modukles.tf.
11. If the operator doesn’t use direct access to Kohl’s network, it is necessary to add the following proxy server to the operator’s environment.
12. Within the test lle folder run below commands:

***terraform init***

***terraform plan -target=module.bootstrap***

***terraform apply -target=module.bootstrap***

**Result: Runner got created**

**4.2.3.2 MODIFY CODEBASE OF CLOUD-ENV-CONFIG**

1. Go to the Repo kship/cloud-env-config.
2. Under the above repo go to the **environments** folder, and to the default file: terraform.tvfars.
3. Make changes specific code to your project and specify the required service accounts.
4. Code changes on the shared folder:
5. Go to the below location. The path is as follows:
6. shared folder-🡪terraform🡪app-roles🡪config.tf, add the service accounts under:
7. variable "app\_roles"variable "app \_vault\_readers"
8. In terraform.tfvars : app\_service\_accounts
9. These changes will trigger a Jenkins job on the respective dev, prod or feature branches and will give an artifact version ID which we will use it while creating a bootstrap environment.
10. Jenkins Build JOB running procedure.
11. Go to the Jenkins, using the URL:

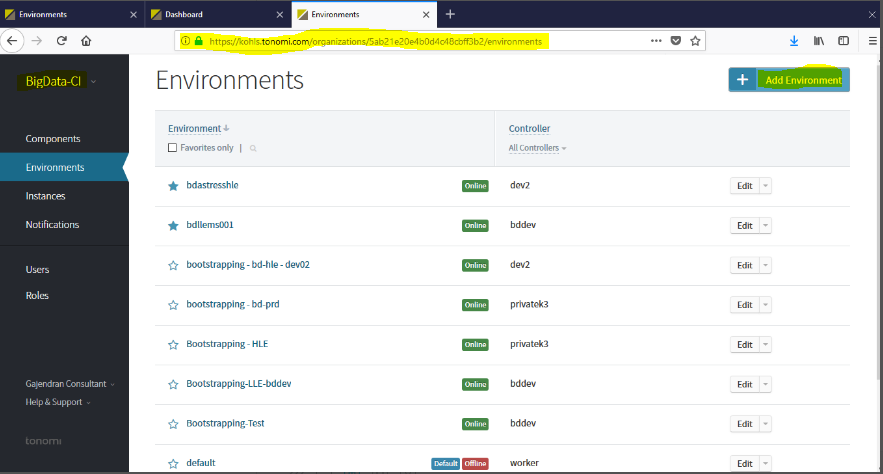
[**https://cicd-jenkins.kohls.com:8443/jenkins/login?from=%2Fjenkins%2F**](https://cicd-jenkins.kohls.com:8443/jenkins/login?from=%2Fjenkins%2F)

1. Search for Kship and then to cloud-env-config. We will see the Below window:
2. Go to the cloud-env-config\_fb\_ build job. Refer to the below screenshot:
3. Now once the job is succeeded, copy the build Number. We will use this build number while add properties to our Bootstrapping environment, Under the cloud-env-config version.

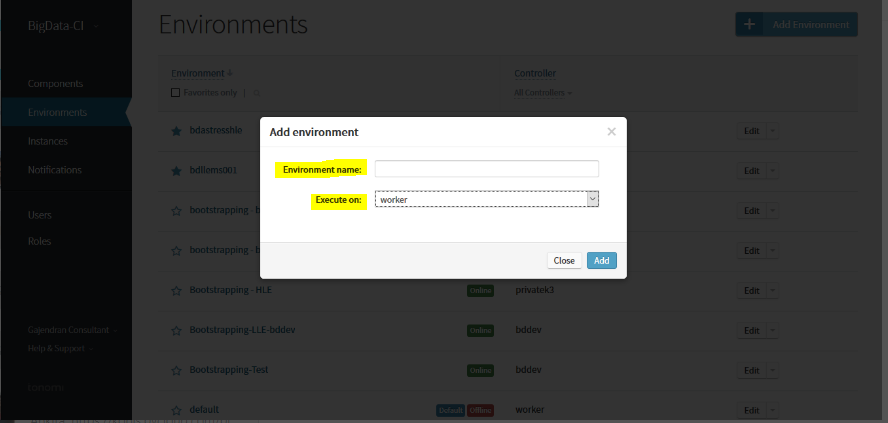
**4.2.3.3 BOOTSTRAPPING ENVIRONMENT**

**Steps for Boot strapping environment:**

1. Go to the Tonomi URL, Choose the BigData-CI Organization and click On Add-Environment.



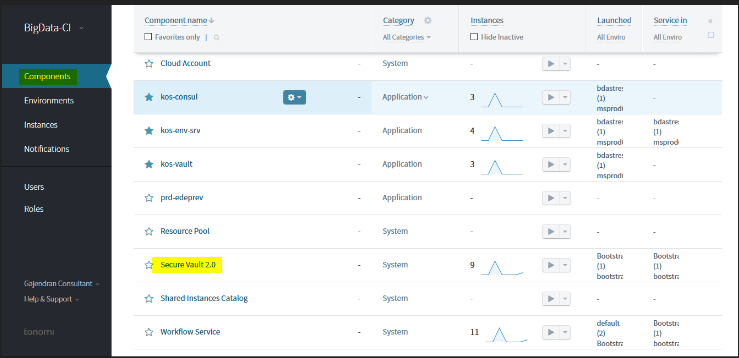
1. Give an Appropriate Name for the environment and give the name of the controller which we want for the bootstrapping environment.



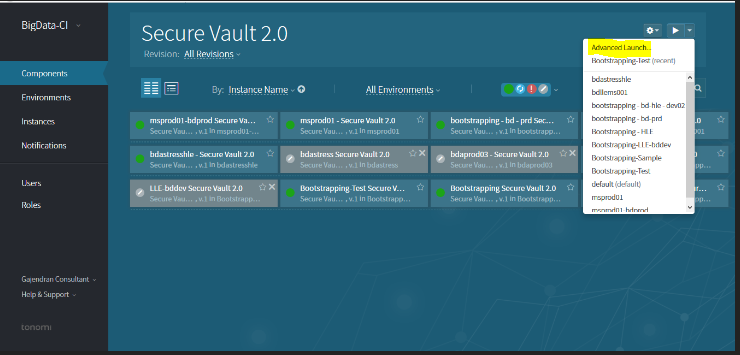
1. Click on the newly created bootstrapped environment which will give the below window. Here we have to add services, workflow policy, and add a property.



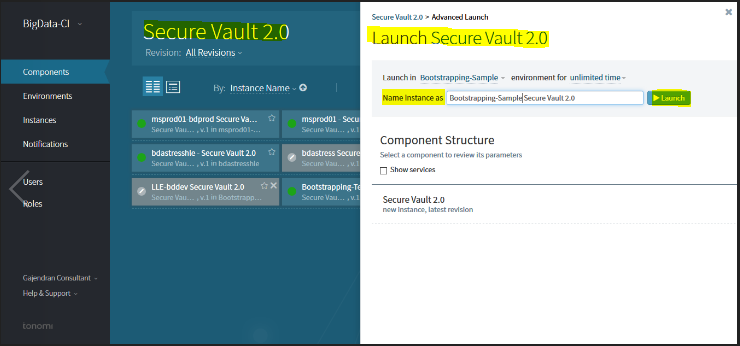
1. Click on the components on the top left-hand side corner and select secure Vault2.0 which will give the below window.



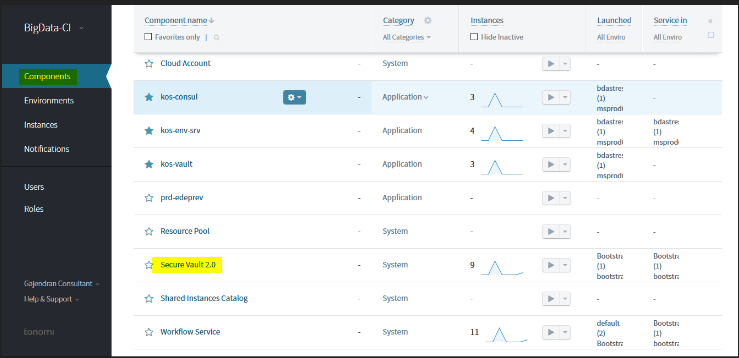
1. On This window, choose the option as shown in figure and choose the advanced launch option and choose the newly created bootstrap environment.



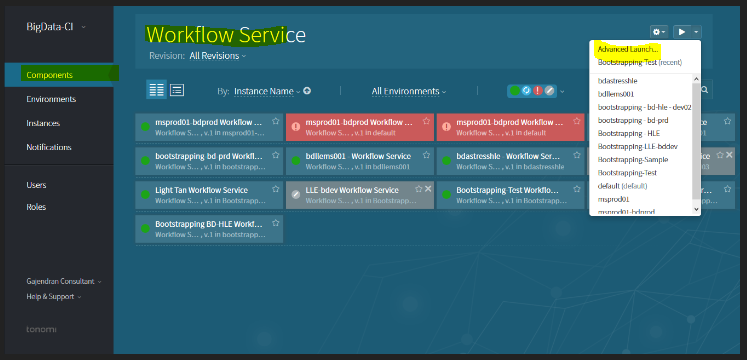
1. Now give a suitable name to secure Vault instance and click on Launch.



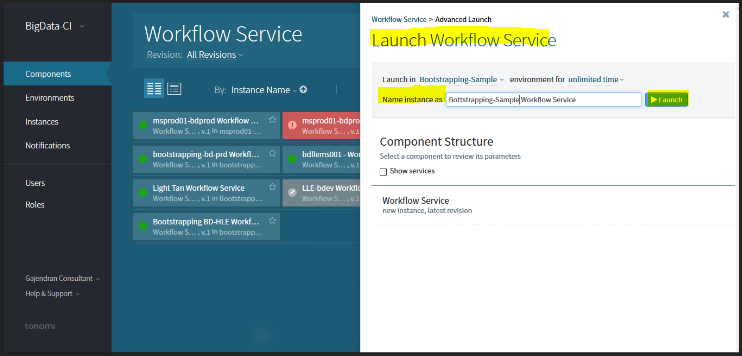
1. Now go back to the components and choose Workflow service.



1. In this window choose the advanced launch option and choose the newly created bootstrapped environment.

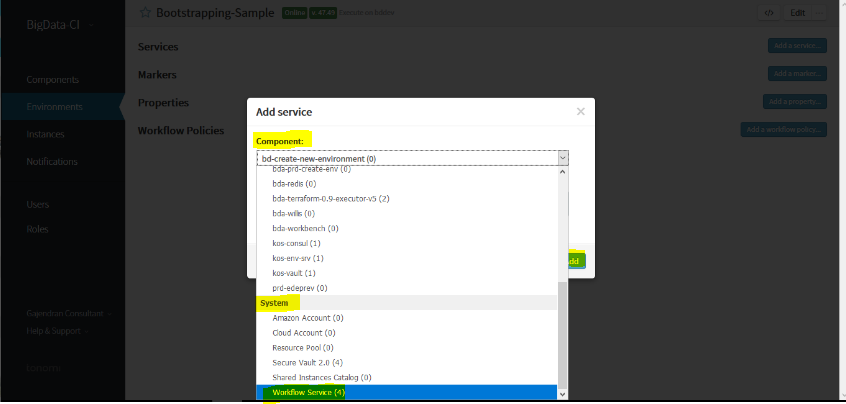


1. Give a suitable name to the instance and click on Launch.

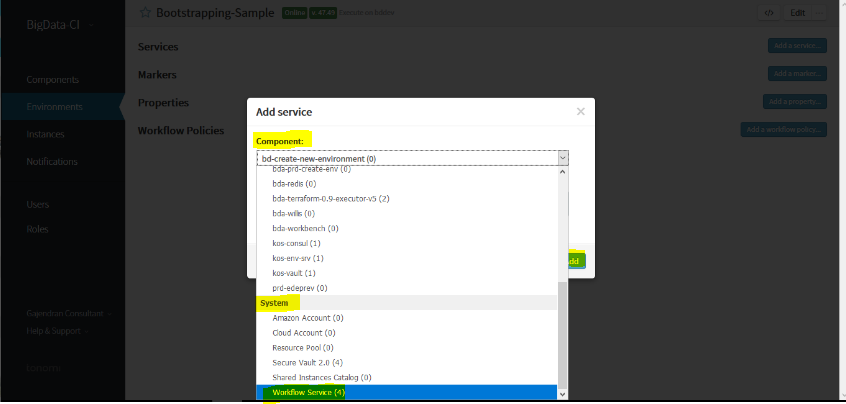


1. Now the two components Secure vault and Workflow service are already created
2. We need to add these two components under the service as shown below:
3. Go back to environments and select the newly created environment.
4. Click on add a service.
5. Choose the newly created bootstrapped environment under the Component,

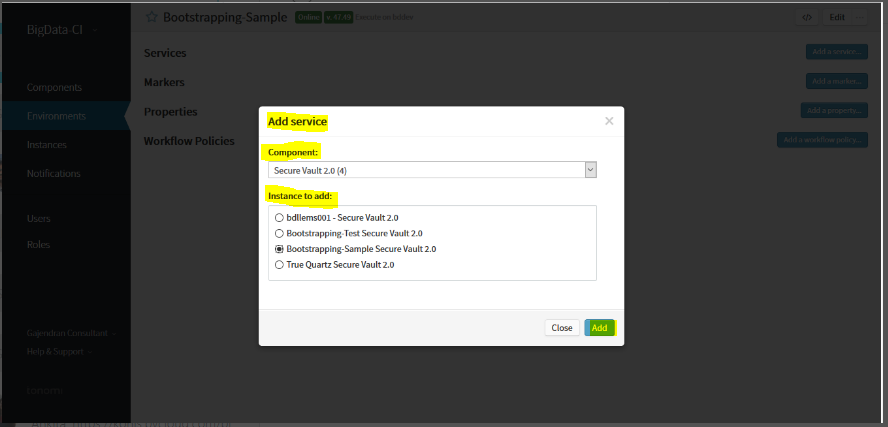
and choose secure vault 2.0 and click on add.



1. Choose the newly created bootstrapped environment under the Component , choose on workflow service and click on add.

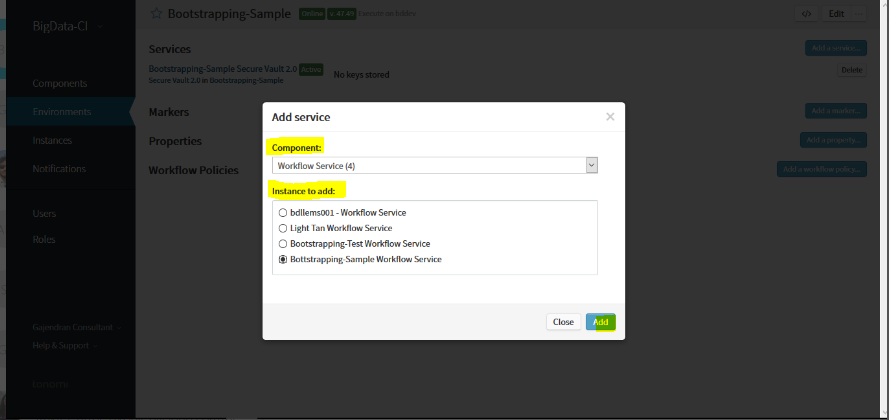


1. Choose the secure Vault 2.0 under the component and under the instance to add give the name of the secure vault instance that was added when the component was created.

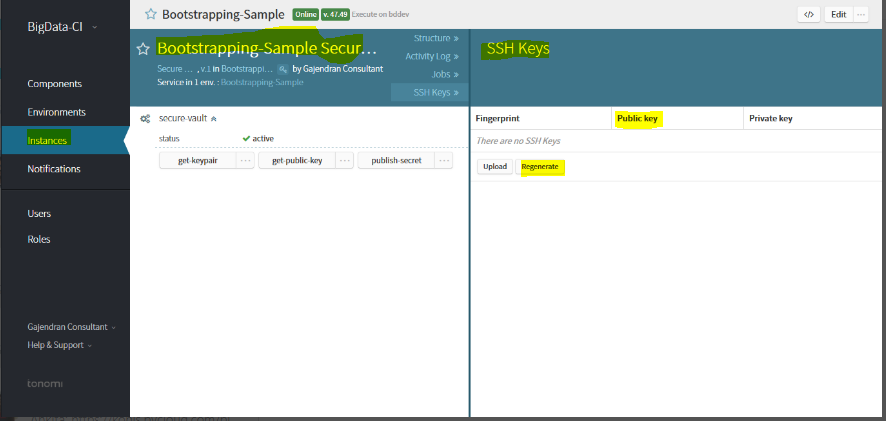


1. Choose the workflow service under the component and under the instance to add, give

the name of the workflow instance that was added when the component was created.



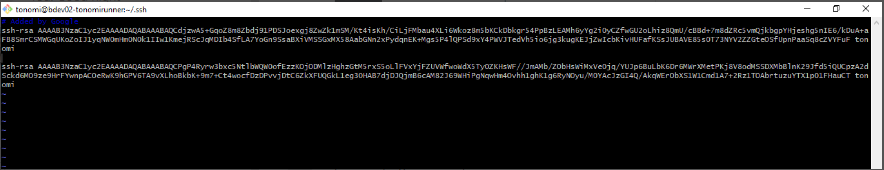
1. Time to login to the runner instance, we need the generate the public keys for this, Follow the steps:



1. Now let’s ssh to the runner instance, by following the below steps:

* ***root user: sudo su***
* ***Login as tonomi user using the command: su tonomi.***
* ***Go too .ssh directory,***
* ***commands: cd ..***
* ***cd .ssh***
* ***vi authorizedkeys***

1. Give the public key or paste the public key that was generated and append "tonomi" as user at the end of the shh key that was generated.



1. In the Properties, click on add property, give the name, type and value and click on add.
2. The fields that we are specifying in the properties are as follows:

***ip:(IP of the runner)***

***user:(username of the tonomi)***

***controller being used:()***

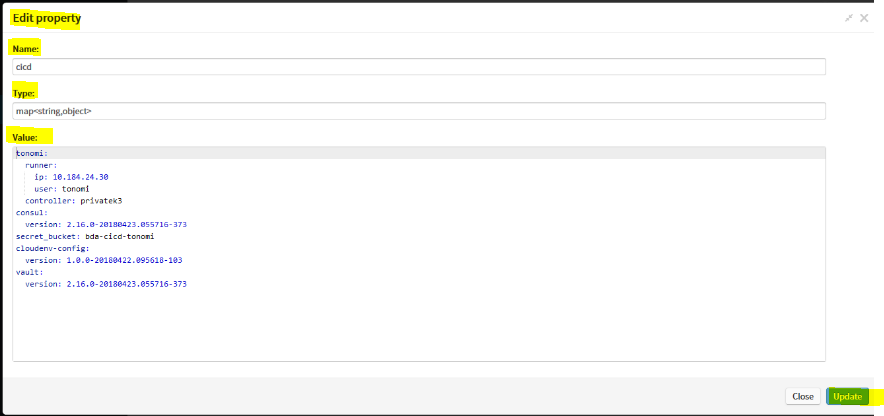
***secret bucket :(Name of the secret bucket)***

***cloud-env-config:***

***version: (Build Number from Jenkins Build job)***

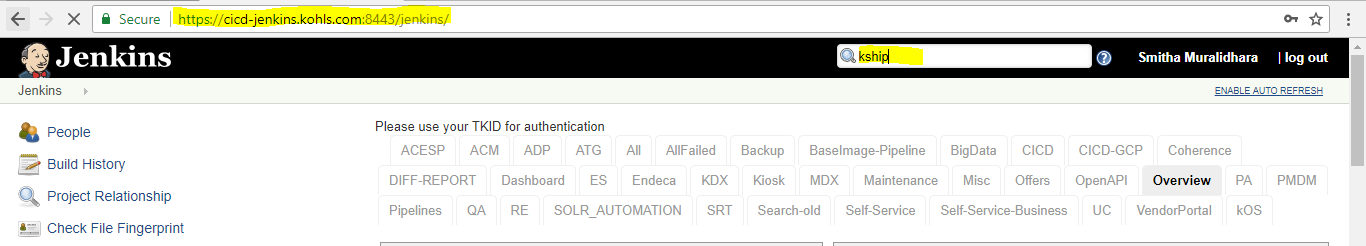
***Vault***

***version:***



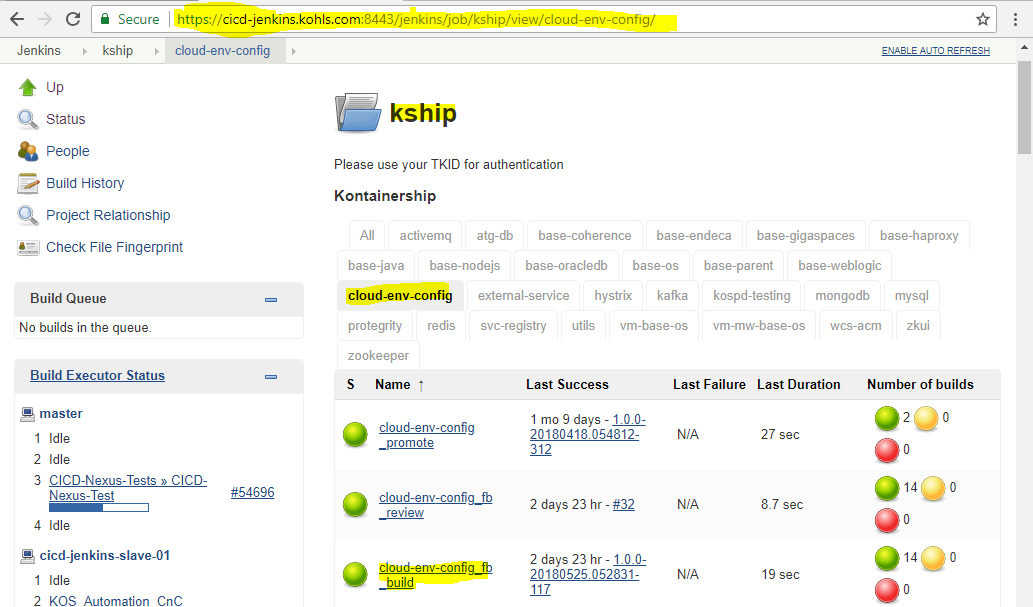
**Steps to get the artifact version from Jenkins:**

1. Go to Jenkins using the URL : [**https://cicd-jenkins.kohls.com:8443/jenkins/**](https://cicd-jenkins.kohls.com:8443/jenkins/)and search for kship as shown below:



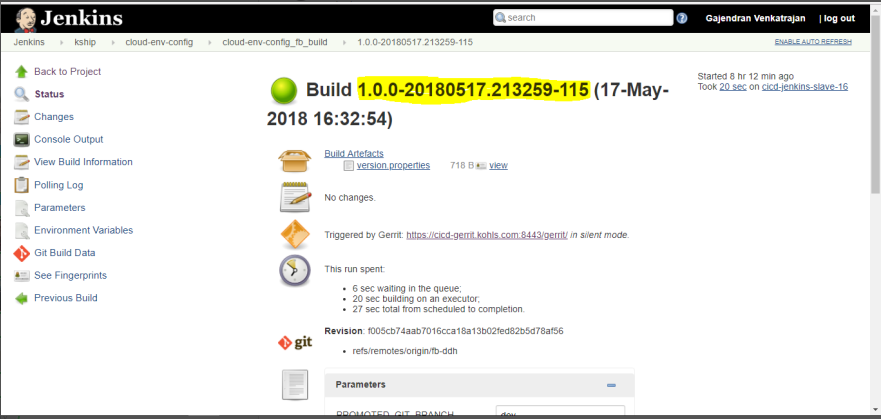
1. Go to the kship project and select the cloud-env-config, and select the build job either on the (dev,

feature or the release) branch as per requirement

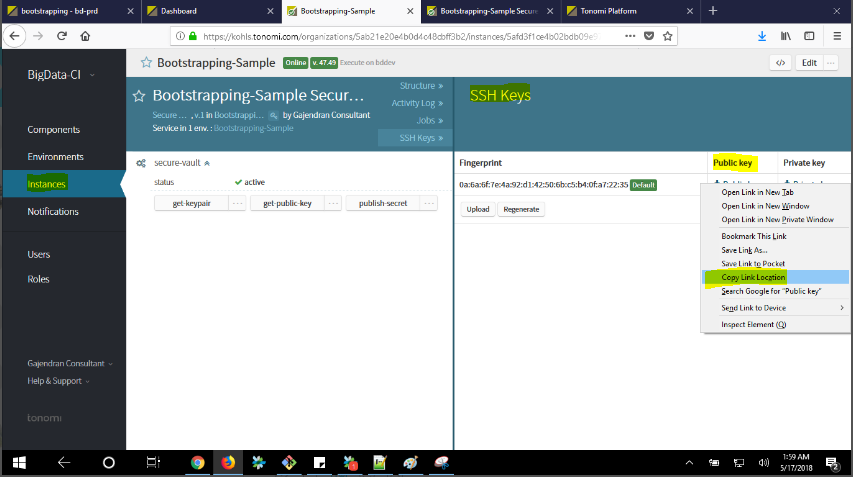


* + - 1. Once the build job runs successfully a build number will be generated, which will be as follows:

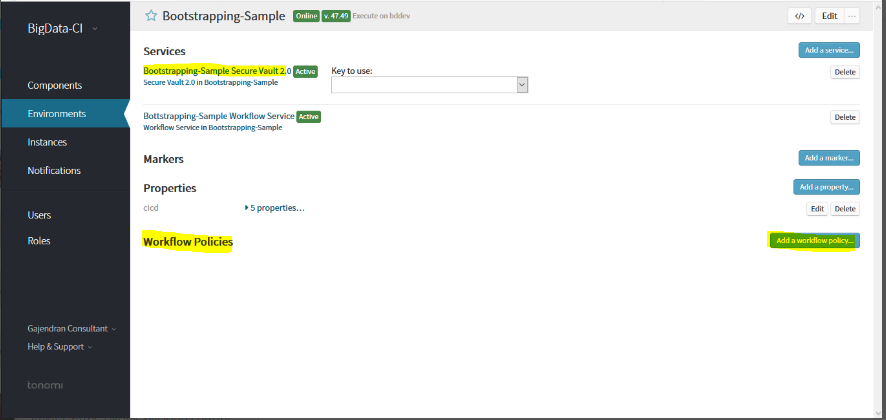
4. This Build number will be used in creation of bootstrapping environment.



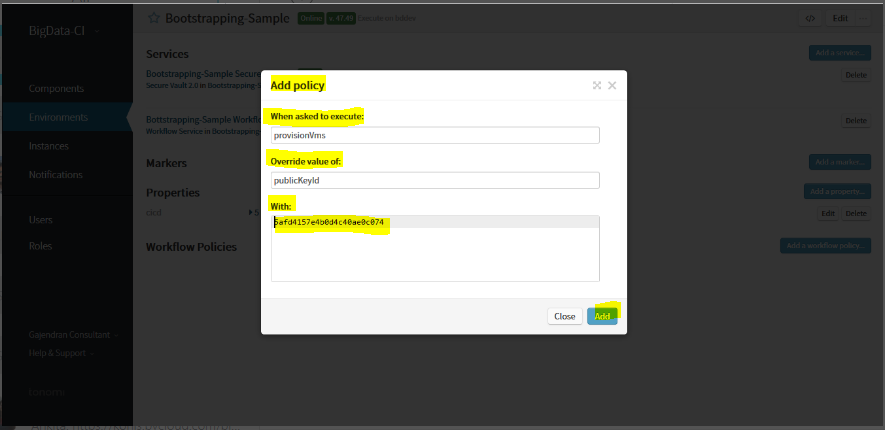
1. Now we have to add the workflow policy, for this we follow the below steps:
2. Go to the secure vault instance that was created and copy the link location of the public key. Save the ID.
3. Once the ID is saved, now come back to the environments and click on the newly bootstrapped environment.



1. Click on add a workflow policy, fill in the all the required details for:



1. when asked to execute:
2. Override Value of:
3. With:
4. Replace it with the ID stored in the previous step and click on ADD.

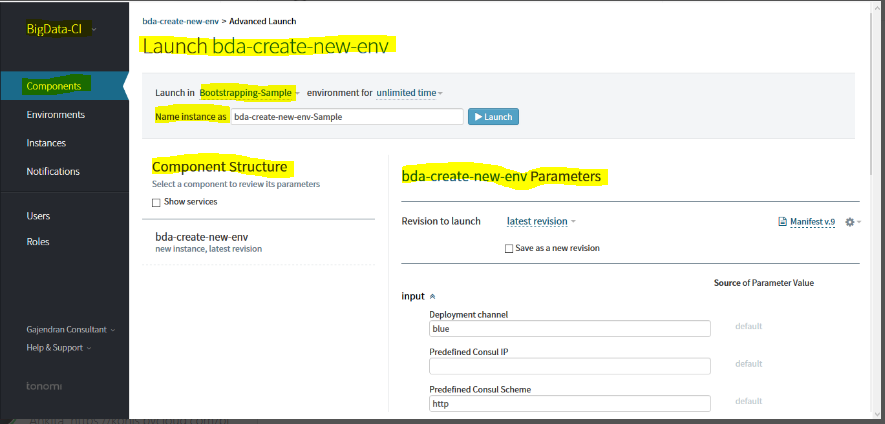


#### 4.2.3.4 Environment Creation:

1. Go to the Tonomi using the URL: <https://kohls.tonomi.com/signIn>
2. Click on the components and select the component bda-create-new-env.
3. In the Launch in tab, mention the name of the bootstrapping environment which was

created previously, select the amount of time for which we need this environment.

1. Give a suitable name for environment instance as shown in the below screenshot:



5. Under the component structure, give the parameters for the new environment,

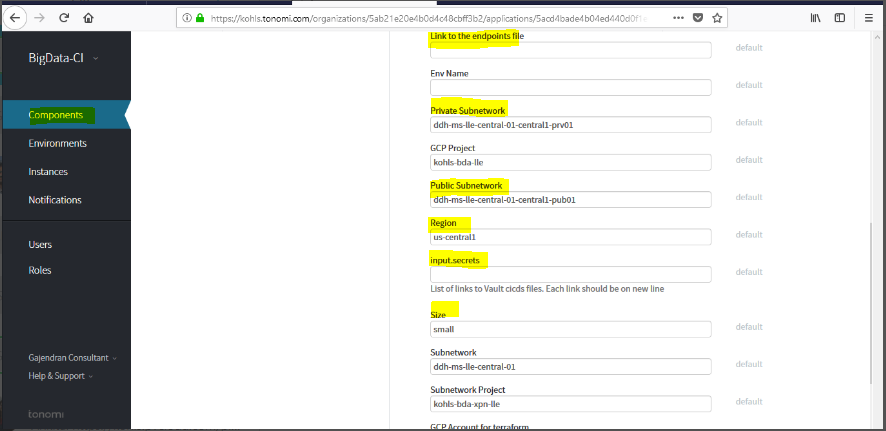
specify all the parameters as below:

1. Give the Private subnetwork, GCP Project, Public Subnetwork, Region as required.
2. Now create a GCS bucket and store the below files:

* ***gs://<gcp-project-name>-secrets/<filename\_endpoints>.json: Will have vault secrets that includes SSL keys***
* ***gs:// <gcp-project-name>-secrets/filename\_endpoints.yaml: Env properties***

1. For the parameters:

* ***link to the enpoints file: gs:// <gcp-project-name>--secrets/bdaprd\_endpoints.yaml***
* ***input.secrets: gs:// <gcp-project-name>--secrets/bdaprd\_endpoints.json:***

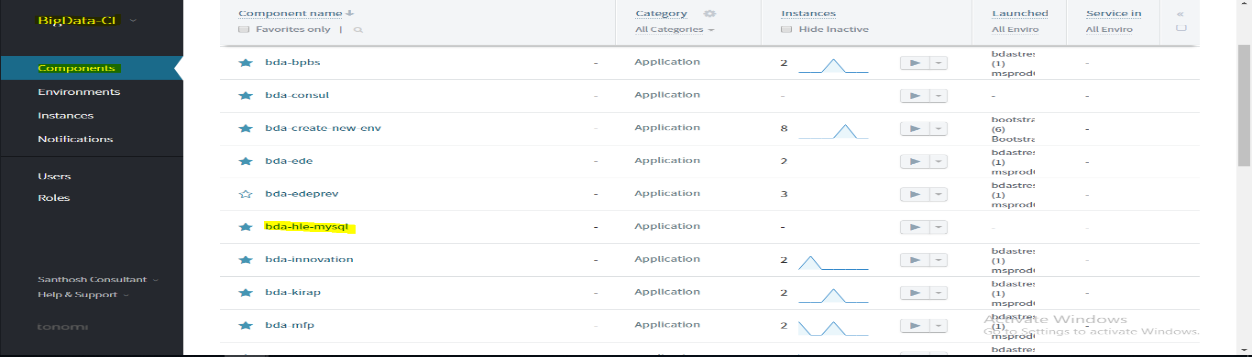


Once all the parameters are given, click on Launch.

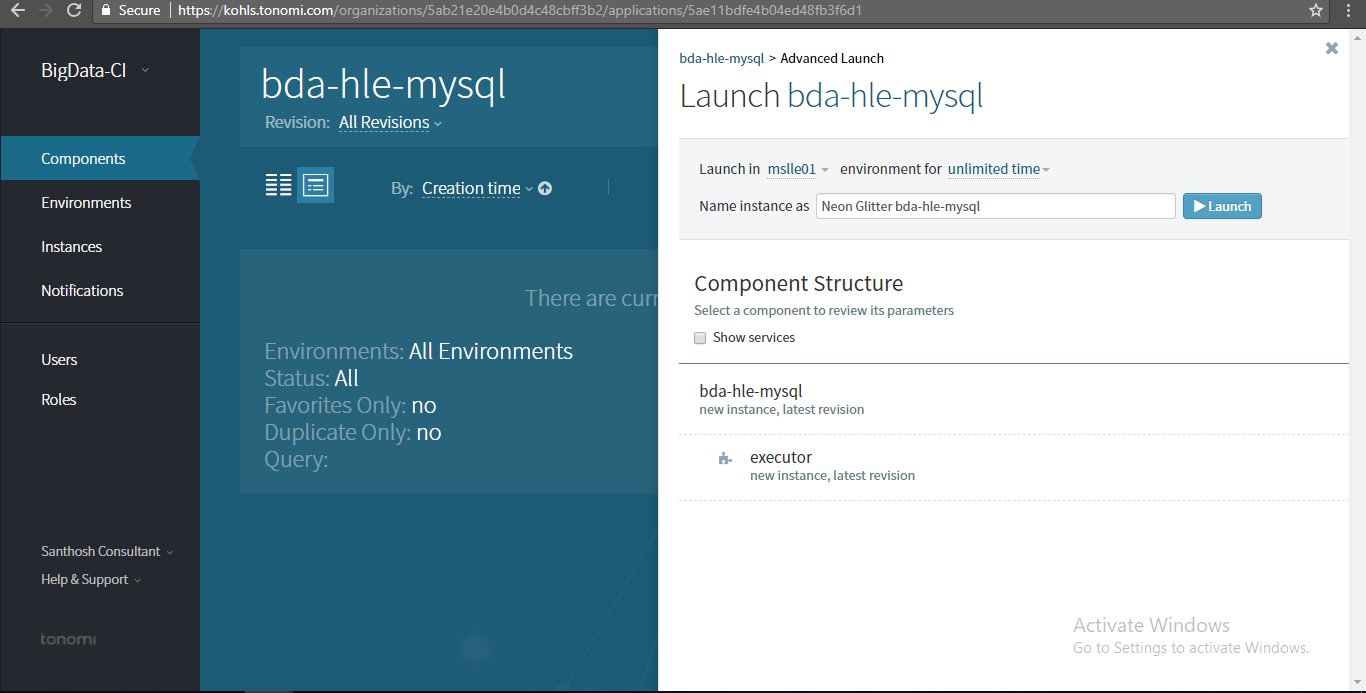
**Result: Environment is created.**

## 4.3 Deployment Steps on Tonomi:

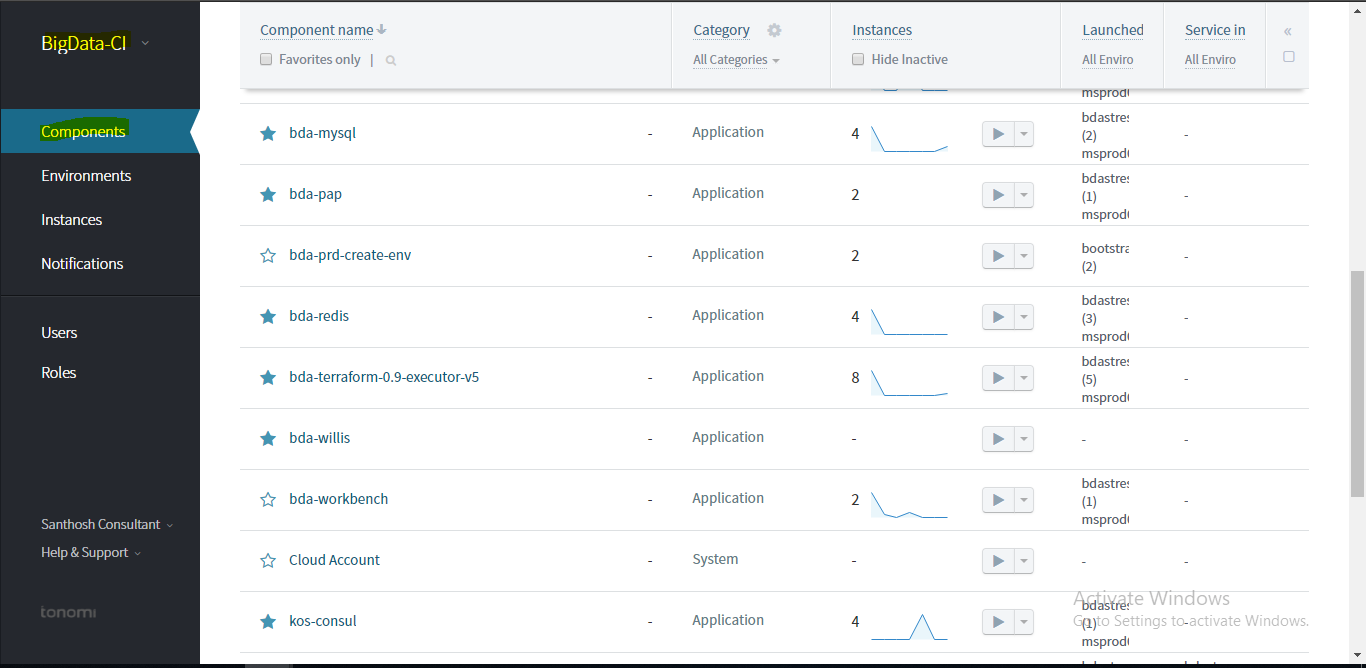
1. Now we have our environment created, we will have to deploy Redis and MySQL from components on Tonomi.
2. For deploying MySQL, select the required component for Tonomi:



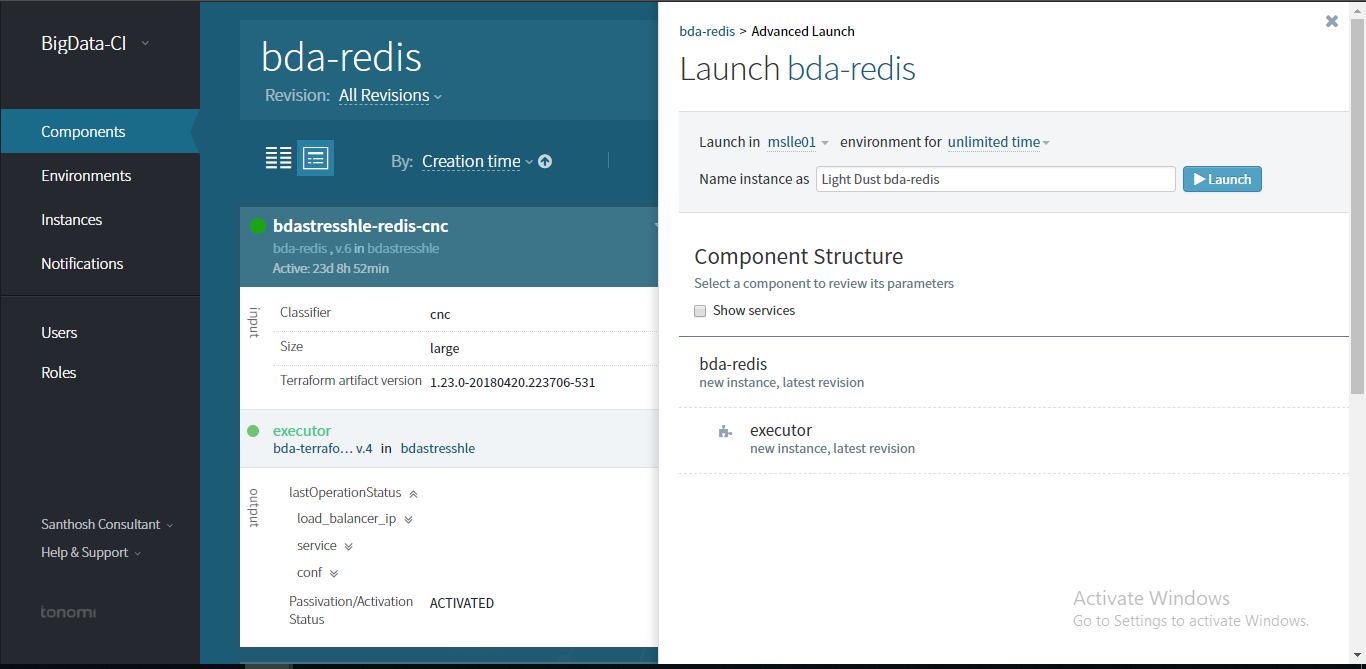
1. Select the SQL component and click on advanced launch, give an appropriate name, an appropriate launch environment and for unlimited time and click on launch.



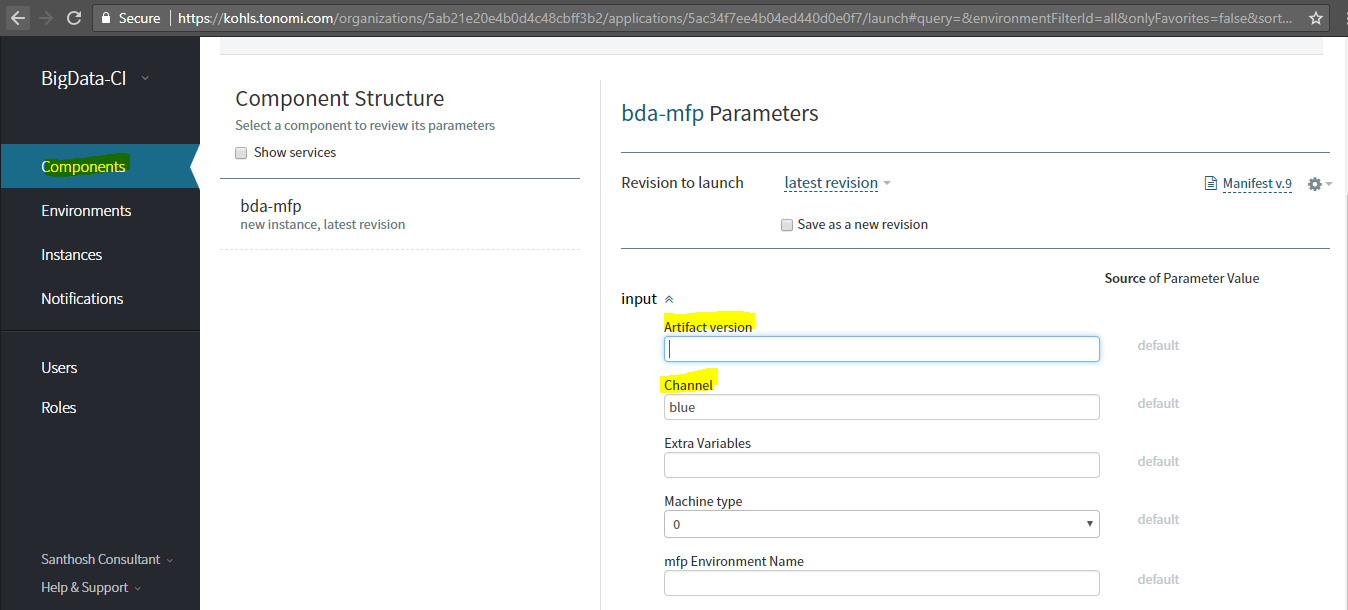
1. Select the Redis component and click on advanced launch, give an appropriate name, an appropriate launch environment and for unlimited time and click on launch.



1. Launch the Redis component in the required environment for unlimited time and give a suitable name for instance and click on Launch.



1. To deploy a microservice, select the pre-existing component from Tonomi, click on the component and select the advanced launch option, give the name of the instance and give all the required details, for **artifact version which will be the build number from Packer job on Jenkins,** Channel is either blue/green. Once all the details are given click on launch.
2. The component will be launched.



**5 Appendix**

**5.1 Gerrit Repo Cloning URL(s)**

|  |  |
| --- | --- |
| **Repo Cloning URL** | **Bigdata Microservice** |
| git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/bigdata-pap  git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/kos/bd-pap | Bigdata-pap  Kos-BD-PAP |
| git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/bigdata-ede-  git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/kos/bd-ede | EDE |
| git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/bigdata-kirap  git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/kos/bd-kirap | Kirap |
| git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/bigdata-bpbs  git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/kos/bd-bpbs | BPBS |
| git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/bigdata-inoapp  git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/kos/bd-inoapp | Inoapp |
| git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/bigdata-mfp  git clone ssh://tkmalve@cicd-gerrit.kohls.com:29418/kos/bd-mfp | MFP |

**5.2 DNS for Bigdata Components:**

**Dev:**

|  |  |  |
| --- | --- | --- |
| **COMPONENT** | **ILB IP** | **DNS** |
| MFP | 10.207.28.52 | [mfp-cloud-ddh-dev.kohls.com](http://mfp-cloud-ddh-dev.kohls.com/) |
| PAP | 10.207.28.45 | [pap-cloud-ddh-dev.kohls.com](http://pap-cloud-ddh-dev.kohls.com/) |
| KIRAP | 10.207.28.59 | [kirap-cloud-ddh-dev.kohls.com](http://kirap-cloud-ddh-dev.kohls.com/) |
| BPBS | 10.207.28.60 | [bpbs-cloud-ddh-dev.kohls.com](http://bpbs-cloud-ddh-dev.kohls.com/) |
| EDE | 10.207.28.63 | [ede-cloud-ddh-dev.kohls.com](http://ede-cloud-ddh-dev.kohls.com/) |
| EDEPREV | 10.207.29.192 | [edeprev-cloud-ddh-dev.kohls.com](http://edeprev-cloud-ddh-dev.kohls.com/) |
| INNOAPP | 10.207.29.194 | [innoapp-cloud-ddh-dev.kohls.com](http://innoapp-cloud-ddh-dev.kohls.com/) |
| WORKBENCH | 10.207.29.195 | [workbench-cloud-ddh-dev.kohls.com](http://workbench-cloud-ddh-dev.kohls.com/) |
| WILLIS | 10.207.28.40 | [willis-cloud-ddh-dev.kohls.com](http://willis-cloud-ddh-dev.kohls.com/) |

**QA:**

|  |  |  |  |
| --- | --- | --- | --- |
| **COMPONENT** | **ILB IP** | **DNS** | **IP Address** |
| MFP | 10.207.28.194 | [mfp-cloud-ddh-qa.kohls.com](http://mfp-cloud-ddh-qa.kohls.com/) | 10.207.28.218 |
| PAP | 10.207.28.41 | [pap-cloud-ddh-qa.kohls.com](http://pap-cloud-ddh-qa.kohls.com/) | 10.207.28.39 |
| KIRAP | 10.207.28.207 | [kirap-cloud-ddh-qa.kohls.com](http://kirap-cloud-ddh-qa.kohls.com/) | 10.207.28.203 |
| BPBS | 10.207.28.206 | [bpbs-cloud-ddh-qa.kohls.com](http://bpbs-cloud-ddh-qa.kohls.com/) | 10.207.28.200 |
| EDE | 35.193.242.193 | [ede-cloud-ddh-qa.kohls.com](http://ede-cloud-ddh-qa.kohls.com/) | 10.207.28.221 |
| EDEPREV | 10.207.29.208 | [edeprev-cloud-ddh-qa.kohls.com](http://edeprev-cloud-ddh-qa.kohls.com/) | 10.207.28.22 |
| INNOAPP | 10.207.28.205 | [innoapp-cloud-ddh-qa.kohls.com](http://innoapp-cloud-ddh-qa.kohls.com/) | 10.207.28.19 |
| WORKBENCH | 10.207.28.54 | [workbench-cloud-ddh-qa.kohls.com](http://workbench-cloud-ddh-qa.kohls.com/) | 10.207.28.192 |
| WILLIS |  | [willis-cloud-ddh-qa.kohls.com](http://willis-cloud-ddh-qa.kohls.com/) |  |

**Stress:**

|  |  |  |  |
| --- | --- | --- | --- |
| **COMPONENT** | **ILB IP** | **DNS** | **SSL** |
| MFP | 10.186.82.6 | [mfp-cloud-bda-hle.kohls.com](http://mfp-cloud-bda-hle.kohls.com/) | Done |
| PAP | 10.186.82.11 | [pap-cloud-bda-hle.kohls.com](http://pap-cloud-bda-hle.kohls.com/) | Done |
| KIRAP | 10.186.82.5 | [kirap-cloud-bda-hle.kohls.com](http://kirap-cloud-bda-hle.kohls.com/) | Done |
| BPBS | 10.186.82.13 | [bpbs-cloud-bda-hle.kohls.com](http://bpbs-cloud-bda-hle.kohls.com/) | Done |
| EDE | 10.186.82.79 | [ede-cloud-bda-hle.kohls.com](http://ede-cloud-bda-hle.kohls.com/) | Done |
| EDEPREV | 10.186.82.9 | [edeprev-cloud-bda-hle.kohls.com](http://edeprev-cloud-bda-hle.kohls.com/) | Done |
| INNOAPP | 10.186.82.7 | [innoapp-cloud-bda-hle.kohls.com](http://innoapp-cloud-bda-hle.kohls.com/) | Pending |
| WORKBENCH | 10.186.82.8 | [workbench-cloud-bda-hle.kohls.com](http://workbench-cloud-bda-hle.kohls.com/) | Pending |
| WILLIS | 10.186.82.12 | [willis-cloud-bda-hle.kohls.com](http://willis-cloud-bda-hle.kohls.com/) | Pending |

**Prod:**

|  |  |  |  |
| --- | --- | --- | --- |
| **COMPONENT** | **ILB IP** | **DNS** | **SSL** |
| MFP | 10.184.24.16 | [mfp-cloud.kohls.com](http://mfp-cloud.kohls.com/) | Done |
| PAP | 10.184.24.13 | [pap-cloud.kohls.com](http://pap-cloud.kohls.com/) | Done |
| KIRAP | 10.184.24.10 | [kirap-cloud.kohls.com](http://kirap-cloud.kohls.com/) | Done |
| BPBS | 10.184.24.15 | [bpbs-cloud.kohls.com](http://bpbs-cloud.kohls.com/) | Done |
| EDE(ELB) | 10.184.24.12 | [ede-cloud.kohls.com](http://ede-cloud.kohls.com/) | Done |
| EDEPREV | 10.184.24.14 | [edeprev-cloud.kohls.com](http://edeprev-cloud.kohls.com/) | Done |
| INNOAPP | 10.184.24.9 | [innoapp-cloud.kohls.com](http://innoapp-cloud.kohls.com/) | Done |
| WORKBENCH | 10.184.24.11 | [workbench-cloud.kohls.com](http://workbench-cloud.kohls.com/) | Done |
| WILLIS | 10.184.24.17 | [willis-cloud.kohls.com](http://willis-cloud.kohls.com/) | Pending |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **Prod-Green** |  |  |  |
| **COMPONENT** | **ILB IP** | **DNS** | **SSL** |
| MFP | 10.184.24.69 | [mfp-green-cloud.kohls.com](http://mfp-green-cloud.kohls.com/) | Done |
| PAP | 10.184.24.66 | [pap-green-cloud.kohls.com](http://pap-green-cloud.kohls.com/) | Done |
| KIRAP | 10.184.24.67 | [kirap-green-cloud.kohls.com](http://kirap-green-cloud.kohls.com/) | Done |
| BPBS | 10.184.24.85 | [bpbs-green-cloud.kohls.com](http://bpbs-green-cloud.kohls.com/) | Done |
| EDE | 10.184.24.86 | [ede-green-cloud.kohls.com](http://ede-green-cloud.kohls.com/) | Done |
| EDEPREV | 10.184.24.87 | [edeprev-green-cloud.kohls.com](http://edeprev-green-cloud.kohls.com/) | Done |
| INNOAPP | 10.184.24.88 | [innoapp-green-cloud.kohls.com](http://innoapp-green-cloud.kohls.com/) |  |
| WORKBENCH | 10.184.24.89 | [workbench-green-cloud.kohls.com](http://workbench-green-cloud.kohls.com/) |  |
| WILLIS | 10.184.24.90 | [willis-green-cloud.kohls.com](http://willis-green-cloud.kohls.com/) |  |

* 1. **Akamai and ELB URL(s)**

**Prod**

|  |  |
| --- | --- |
| **Akamai URL** | <https://api-bd.kohls.com/v1/ede/experiences?cid=Demo&pgid=Home&plids=Vertical> |
| **ELB URL** | <https://ede-cloud.kohls.com/v1/ede/experiences?cid=Demo&pgid=Home&plids=Vertical> |

# Stress:

|  |  |
| --- | --- |
| **Akamai URL** | [**https://apistress1-bd.kohlsecommerce.com/v1/ede/experiences?cid=Demo&pgid=Home&plids=Vertical**](https://apistress1-bd.kohlsecommerce.com/v1/ede/experiences?cid=Demo&pgid=Home&plids=Vertical) |
| **ELB URL** | <https://ede-cloud-bda-stress.kohls.com/v1/ede/experiences?cid=Demo&pgid=Home&plids=Vertical> |

**5.4** **Bigdata Microservices Component Details On Tonomi:**



**5.5 GCP URL:**

<https://console.cloud.google.com/>

**5.6 Contact Details:**

|  |  |  |
| --- | --- | --- |
| **Resource Name** | **Tools** | **Email ID** |
| Nishant (onsite) | MDX, RTO, Redis, Tableau | nishant.reddy@kohls.com |
| Syed (onsite) | Atscale, GPU, Azkabhan, DDH Cluster, MySQL | [syed.patel@kohls.com](mailto:syed.patel@kohls.com) |
| Rajesh (onsite) | VP/PMDM, OpenShift, DDE | [rajesh.kasaragadda@kohls.com](mailto:rajesh.kasaragadda@kohls.com) |
| Santhosh Kumar | BPBS, KIRAWeb, MFP, Redis, Mosaic | [santhosh.kumar@kohls.com](mailto:santhosh.kumar@kohls.com) |
| Govardhan | PAP, EDE, EDEPREV, MySQL, Tableau, Zabix, Jira | govardhan.reddy@kohls.com |
| Illyaz | Tableau, Zabix, JIRA, Innoapp, Workbench, Cavision Monitoring | mohammediliyazkasnoor.syed@kohls.com |
| Balasekhar Nelli | MDX, RTO, Mosaic | balasekhar.nelli@kohls.com |
| Jayaprakash | VP/PMDM, OpenShift, Mosaic | tkmakru@kohls.com |
| Abhijeet Singh | Mosaic EDE, EDEPREV, KIRAWeb | abhijeetx.singh@kohls.com |
| Surya Prabhakar | Cavision Monitoring, PAP, MFP, BPBS | surya.prabahar@kohls.com |
| Gajendran V | Architecture, OpenShift & Mosaic, DDH Cluster | gajendran.venkatrajan@kohls.com |
| Ankita | Mosaic PAP, EDE, EDEPREV, OpenShift Readiness | tkmakrx@kohls.com |
| Aswin | Mosaic - L3/L4 | aswin.murthy@kohls.com |