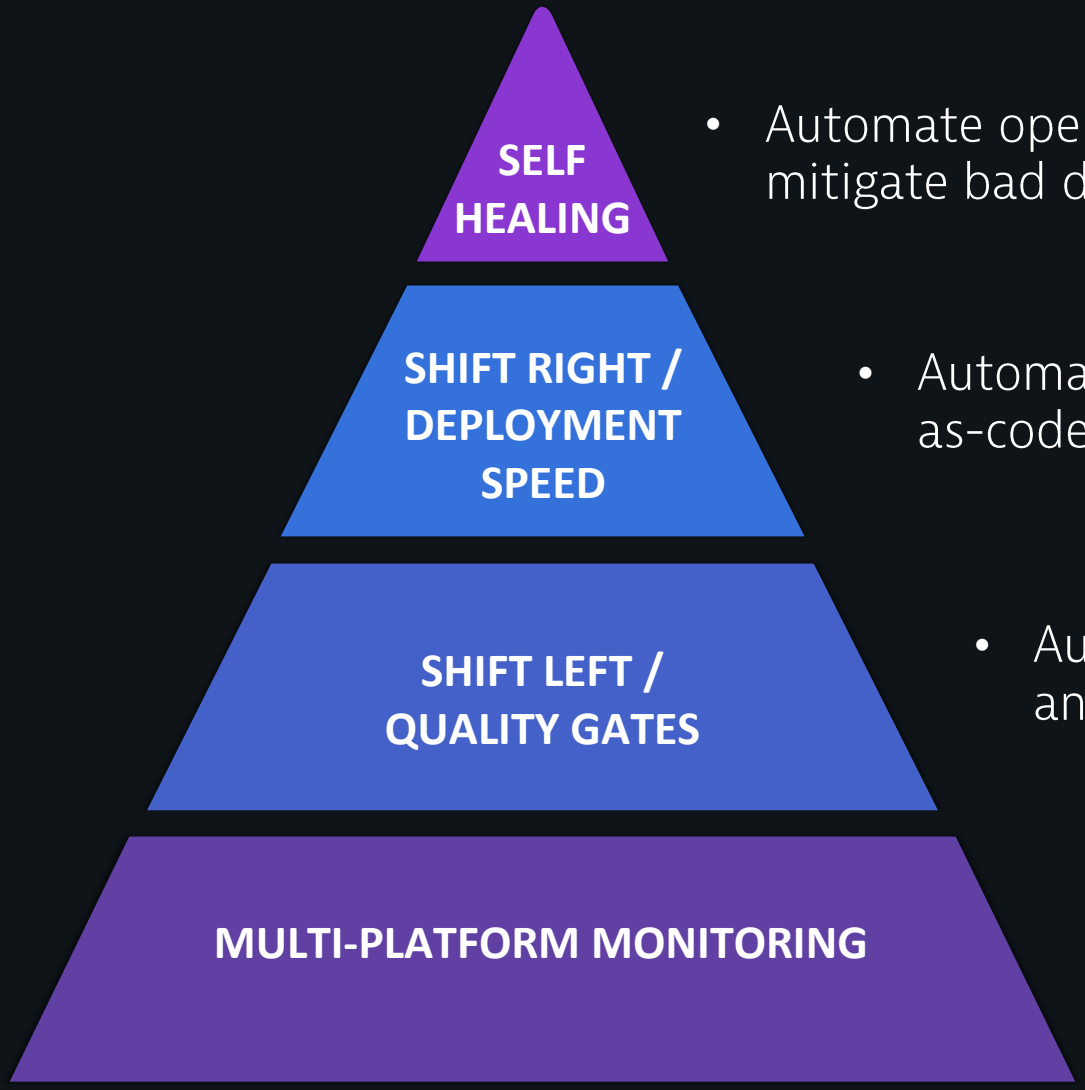
Dynatrace Autonomous Cloud

GKE, Jenkins, Ansible Tower

dynatrace

Key blue print – unbreakable continuous delivery model

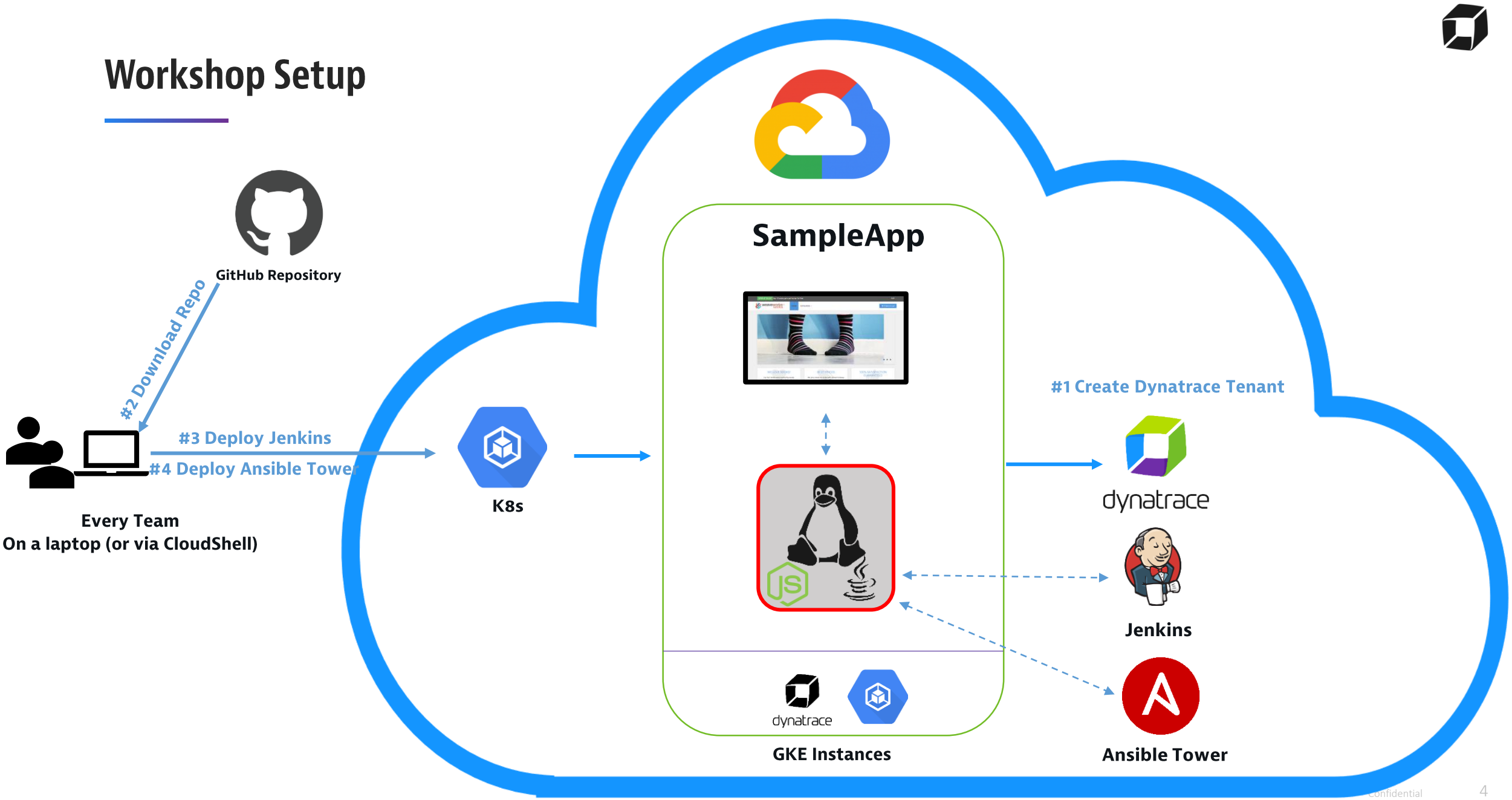


- Automate operations (self-healing) – auto-mitigate bad deployments in production
- Automate deployment (shift-right) – push "monitoring-as-code" for auto-validation and auto-alerting
- Automate quality (shift-left) – automate the pipeline and stop bad code changes before they reach prod
- Automate monitoring – use monitoring strategically as feature of the end-to-end pipeline

What we will accomplish

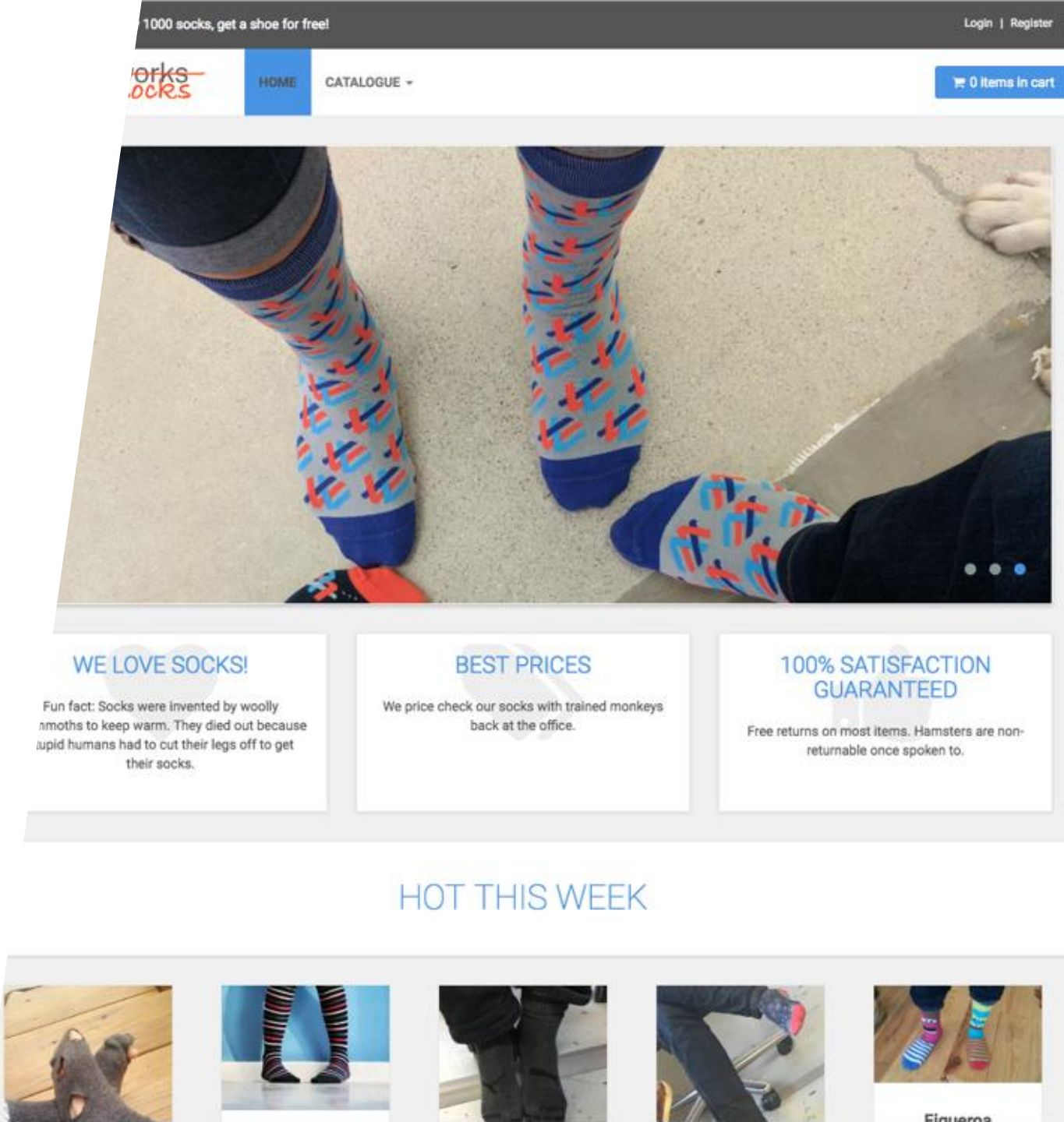
- Deploying the SockShop application to our GKE cluster
- Setting up Jenkins on K8s in GKE
- Import a Jenkins Pipeline
- Setup Dynatrace Integration with Jenkins
  - Using Jenkins Plugin and Monspec
- Deploy different code builds and test their performance
  - Fail pipeline if code is not performant based on monspec
- Create a self-healing application using Dynatrace and Ansible Tower

Workshop Setup

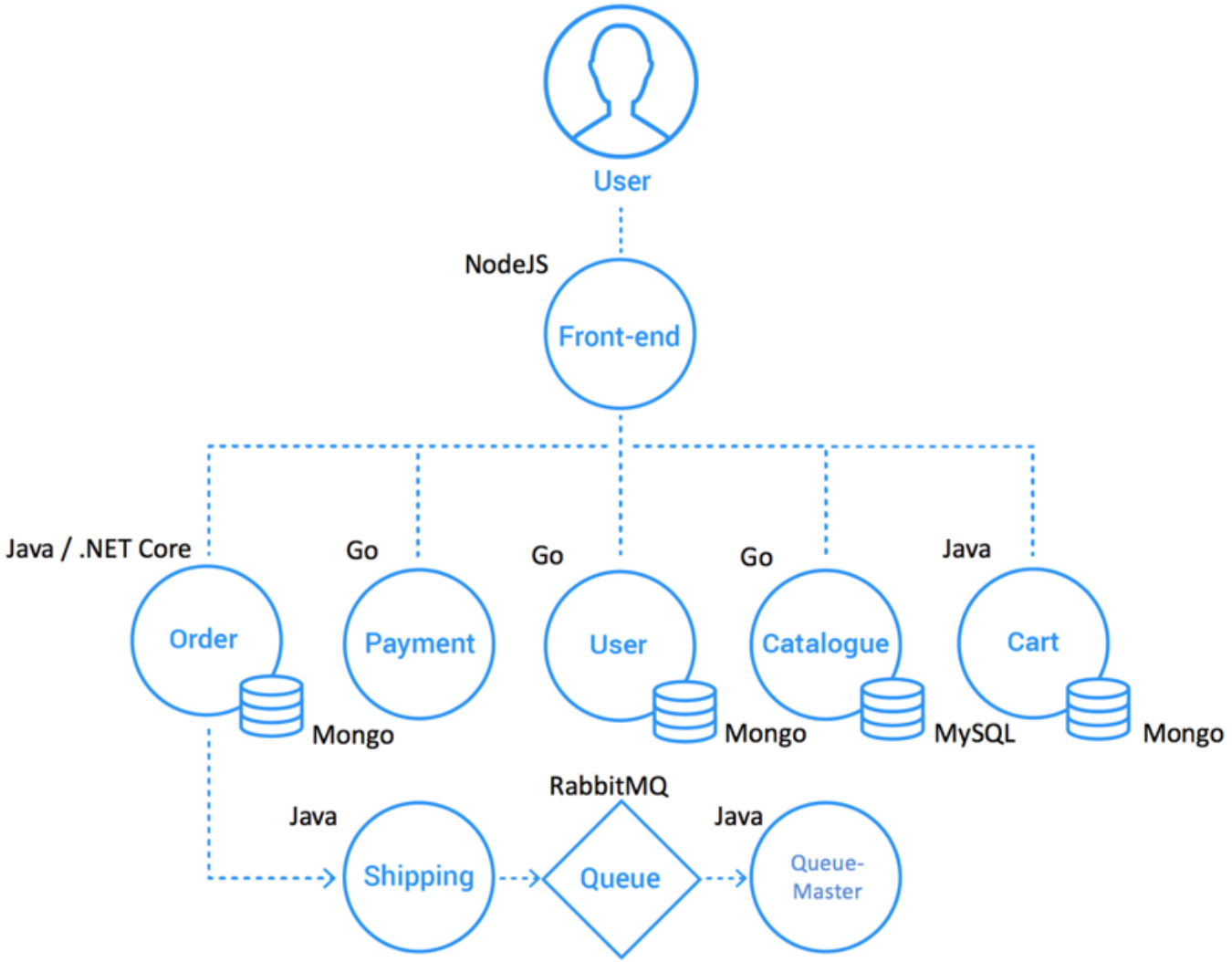


Sock Shop Application

- Based on: http://socks.weave.works
- Opensource microservice sample app
- Polyglot
  - Java Spring Boot
  - Go
  - Node.js
  - RabbitMQ (order queuing)
  - MongoDB, MySQL (data stores)



Sock Shop architecture



Jenkins

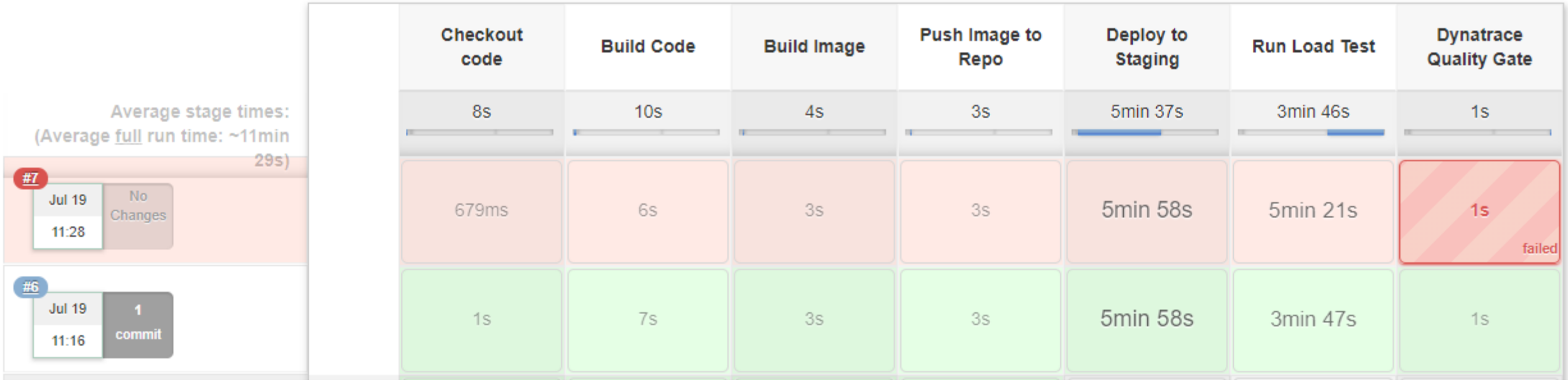
- The Worlds most advanced BASH terminal
- In most cases it's fine to deploy to K8s via simple kubectl commands in k8s
  - Most of deployments probably done this way
  - Spoiler Alert: We're going to be doing this today
- No true "easy" way to determine a failure or if app is up and running

  ```
  $kubectl wait -n dev deploy/myapp --for condition=available
  ```

  This won't catch pods that didn't actually come online

What our Pipeline will look like (hopefully)

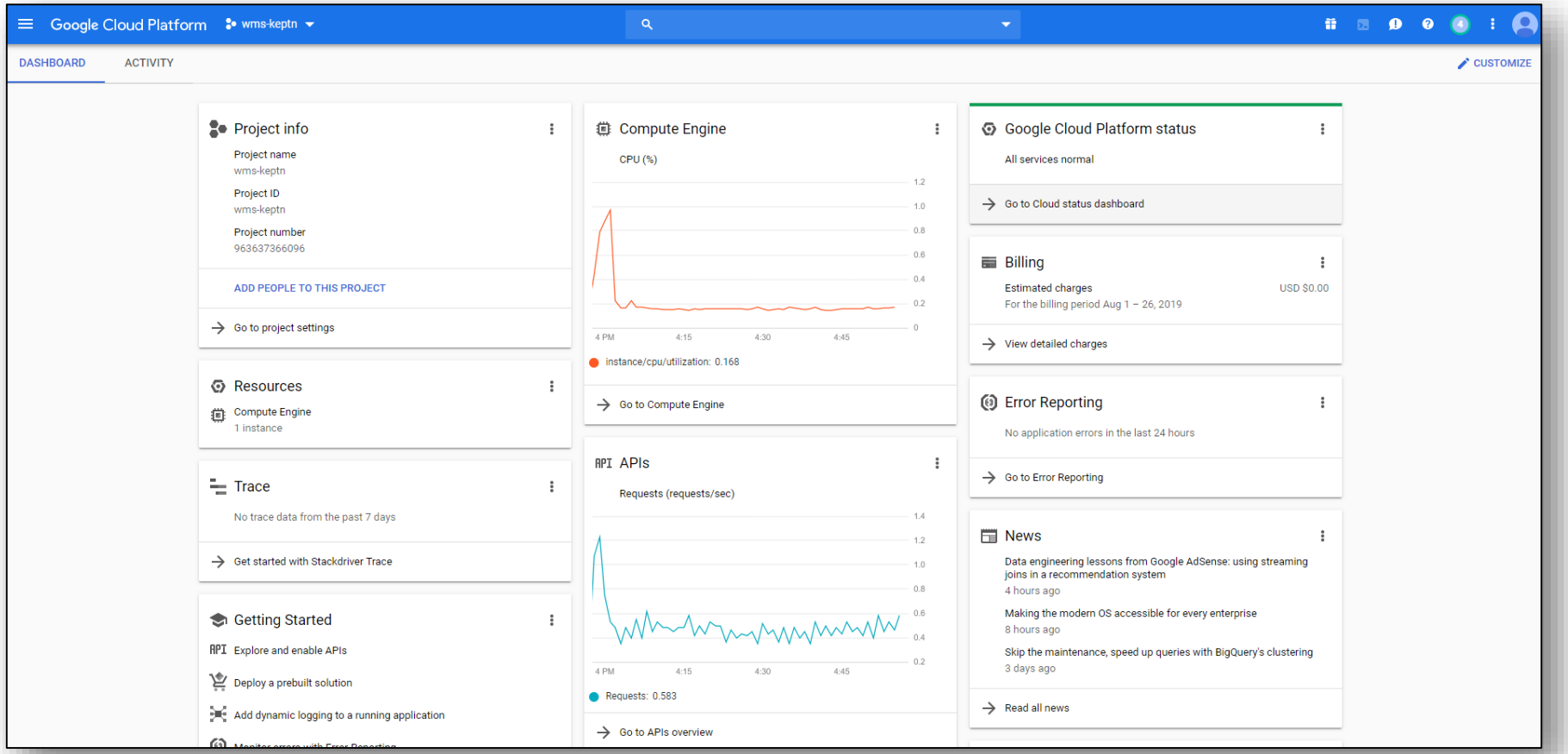
Stage View



Let’s Start the fun

Setup our GKE Cluster

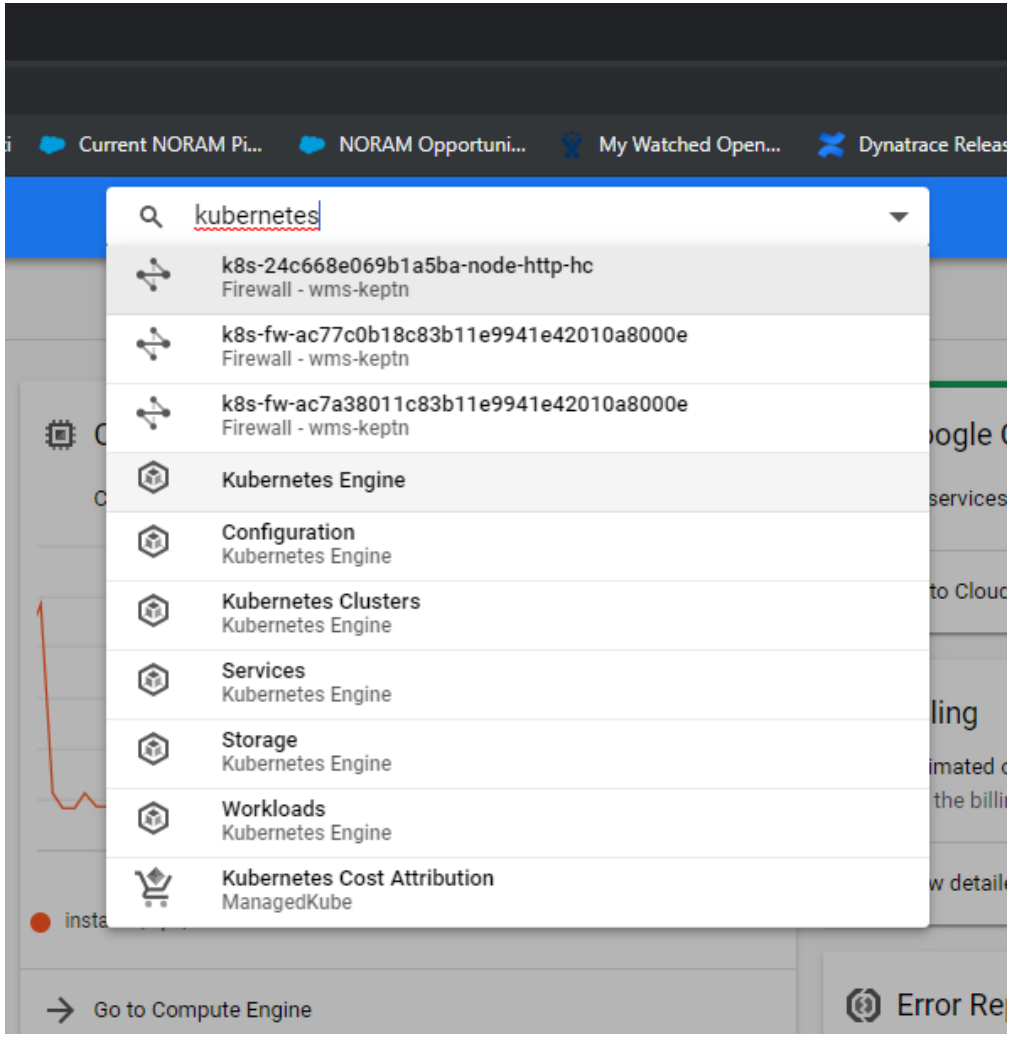
- Login to your GCP Account



Setup our GKE Cluster

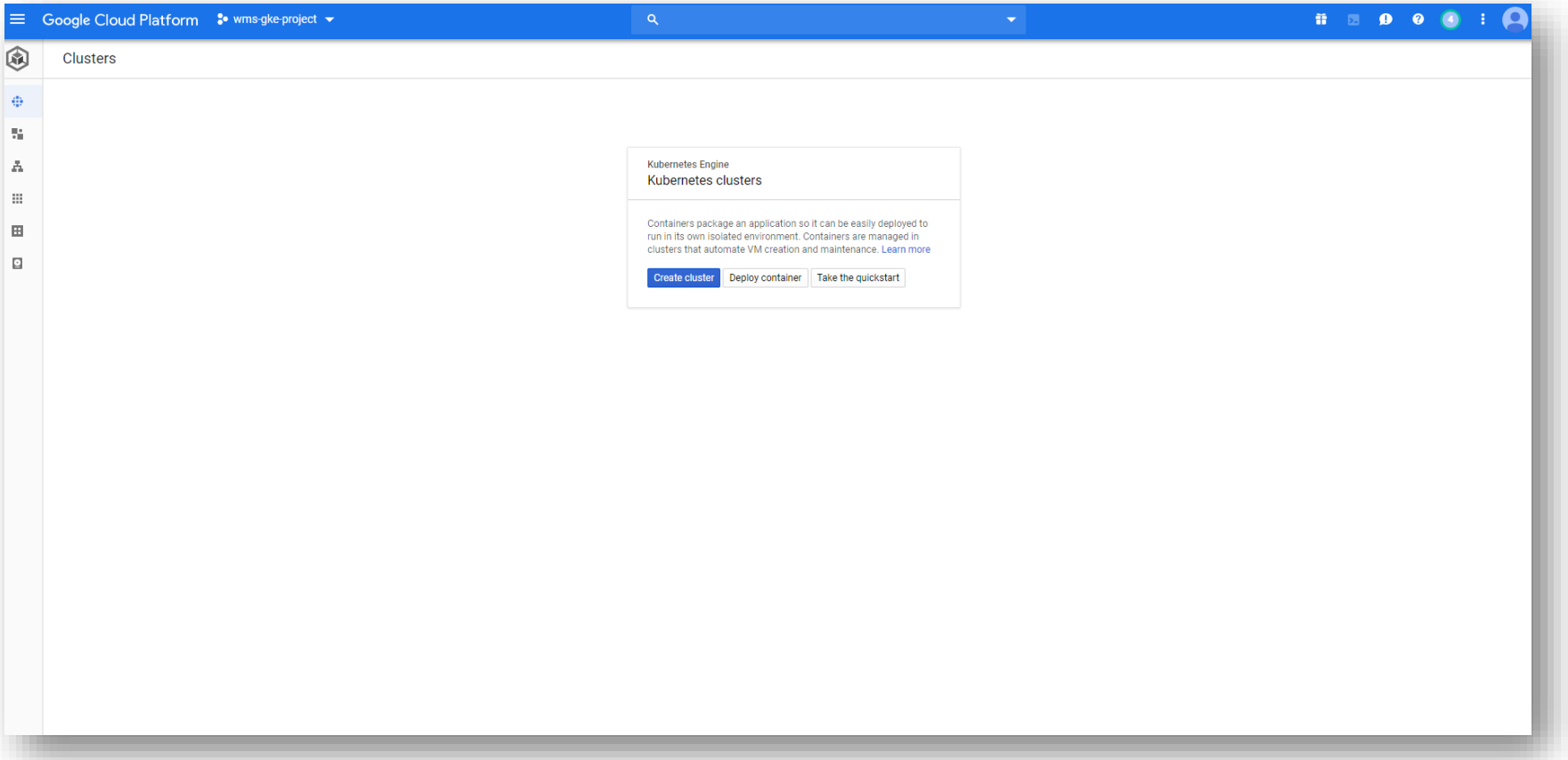
- In the search box, type “Kubernetes” and select “Kubernetes Engine” from the search results

- This may take a few minutes if this is the first time you are attempting to launch Kubernetes in this account.



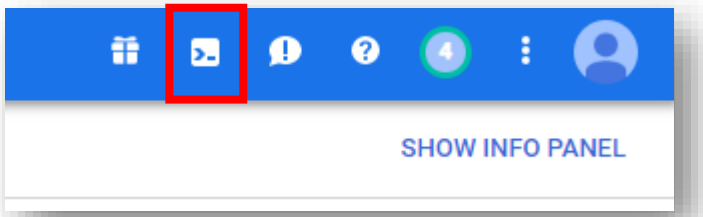
Setup our GKE Cluster

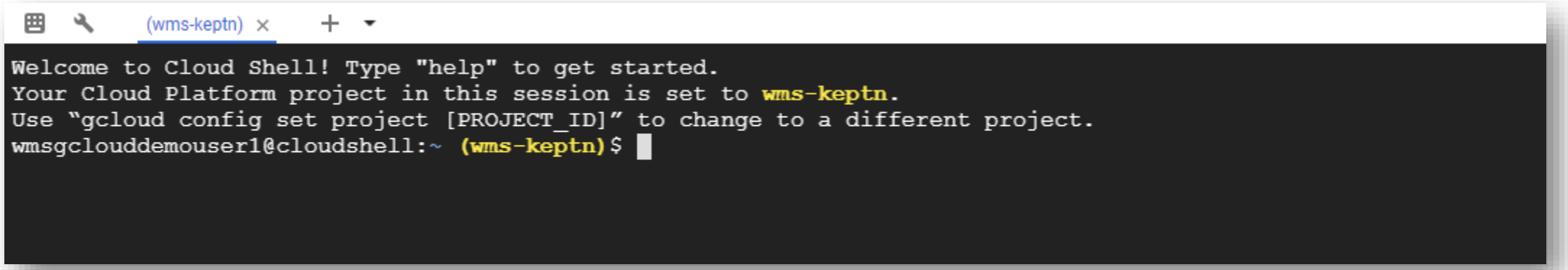
- After the initial APIs are setup, your Kubernetes Engine screen should look like this (unless you already have an existing cluster setup)



Setup our GKE Cluster – Launching Cloudshell

- To launch and interact with our Kubernetes cluster we use Cloudshell
  - Cloudshell provides command-line access to your cloud resources directly from your browser
  - No need for a separate bastion host
  - Required tools pre-installed (git, docker, kubectl, etc...)
  - Persistent storage





Setup our GKE Cluster – Cloning our repository

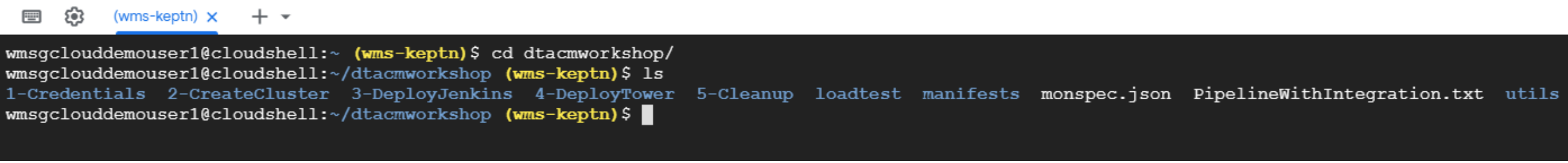
- In Cloudshell, clone/download the following GitHub Repository:

https://github.com/Dynatrace-APAC/workshop-dtac

Run the command: `git clone https://github.com/Dynatrace-APAC/workshop-dtac.git`

- This will create a folder called **workshop-dtac**

- This will contain the scripts we will use for the remainder of the workshop

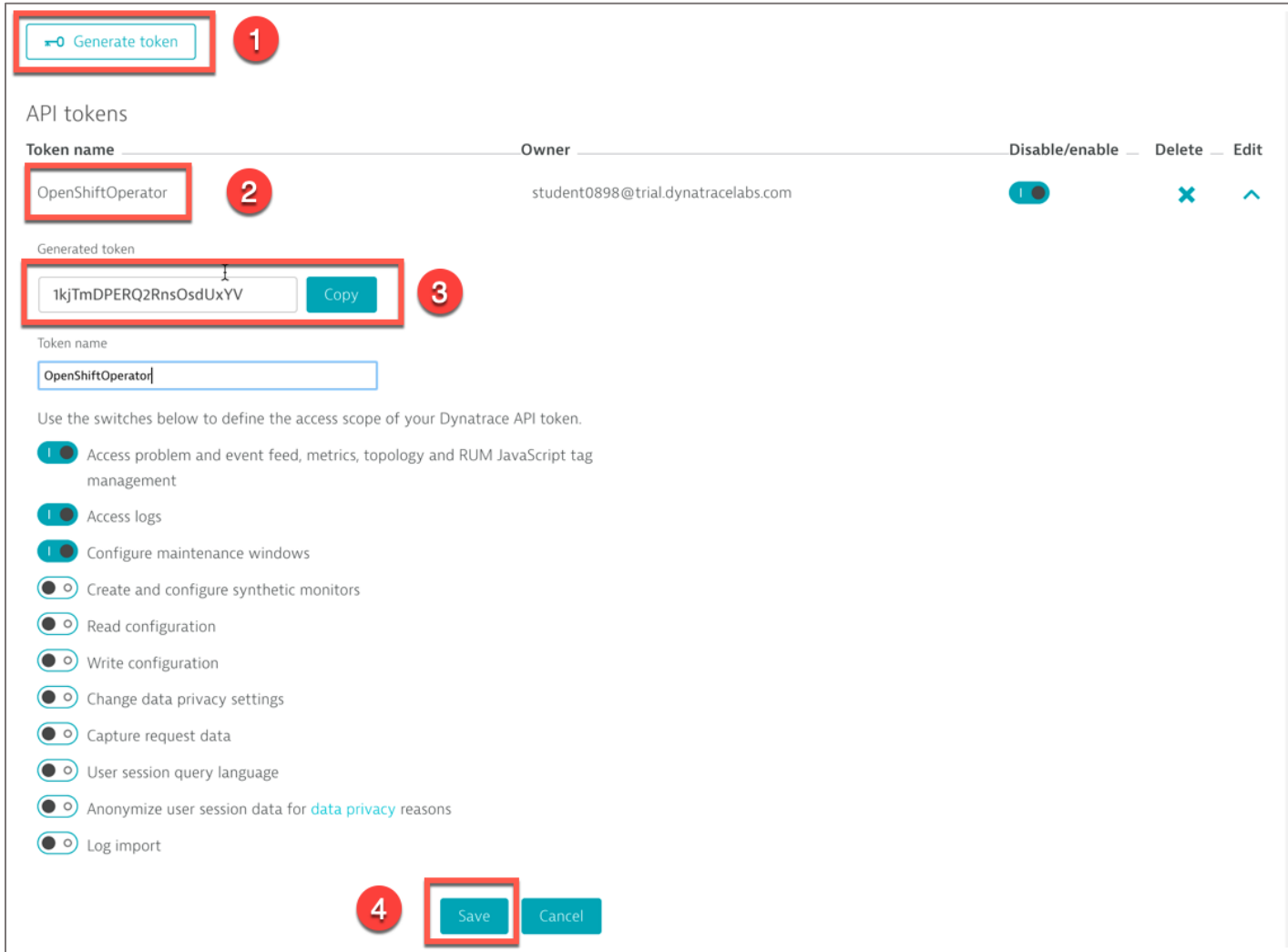


Dynatrace Credentials – Tenant ID & Environment ID

- In order to proceed further we need to make note of our Dynatrace Tenant/Environment and API/PaaS Tokens
- For the Tenant ID, you can find it in the first part of your URL
  - https://<TENANT_ID>.live.dynatrace.com – SaaS Deployments
    - For example, for https://jwx05250.live.dynatrace.com , Tenant ID=jwx05250
  - https://<TENANT_ID>.dynatrace-managed.com
    - For example, for https://abc123.Dynatrace-managed.com, Tenant ID=abc123
- The Environment ID is only applicable for Dynatrace Managed Deployments. You can find this value in the second half of your URL
  - https://<TENANT_ID>.dynatrace-managed.com/e/<ENVIRONMENT_ID>
    - For example, for https://abc1234.Dynatrace-managed.com/e/1234-5678, Environment ID=1234-5678

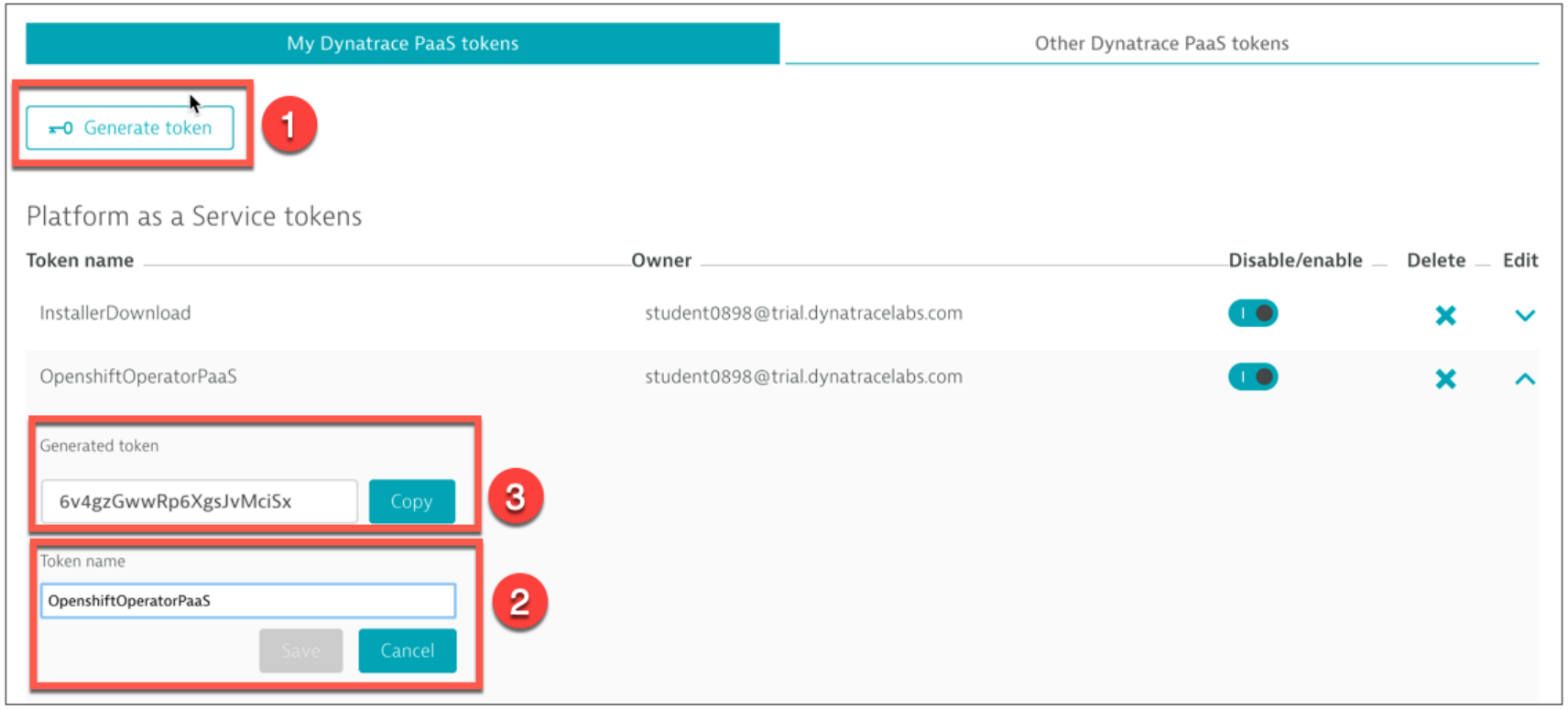
Dynatrace Credentials – API Token

- Go in Settings -> Integration -> Dynatrace API
  1. Click on Generate Token
  2. Enter a name for your token (e.g. GKE)
  3. Enable all the access scope
  4. Don't forget to click on the Save button



Dynatrace Credentials – PaaS Token

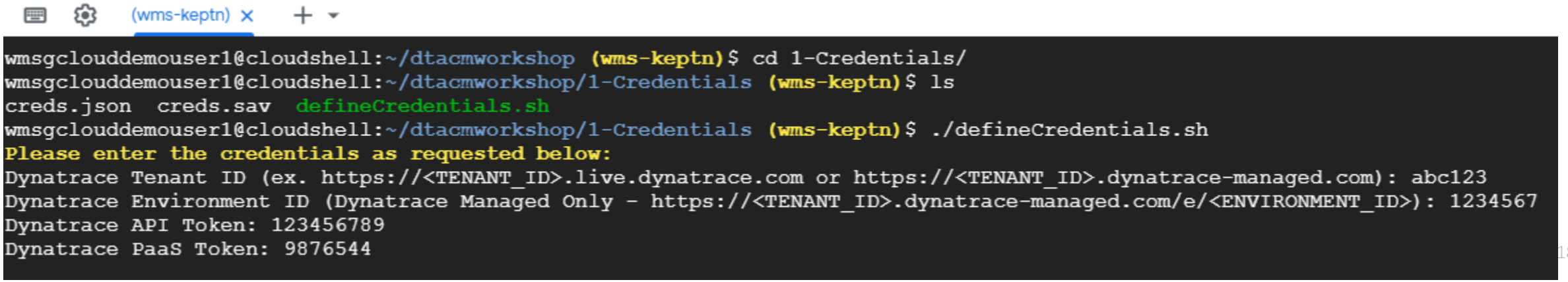
- Go in Settings -> Integration -> Platform as a Service
  1. Either copy the existing InstallerDownload token or click on Generate Token
  2. Enter a name for your token (e.g. GKE), click Save



Creating our Credentials

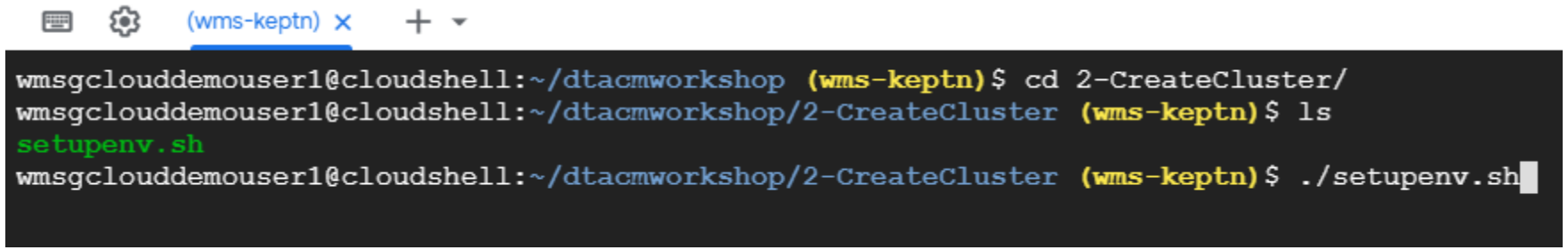
- We now need to store a local copy of our Dynatrace Tenant and API info for use when creating our cluster

- Navigate to the directory **1-Credentials** and execute the script **defineCredentials.sh**
  - This will ask for your Dynatrace Tenant/Environment information as well as your API and PaaS tokens
    - These values can be obtained from your Dynatrace tenant (see previous slides)
  - Once you have entered the values and confirmed they are correct, we can move on to creating our cluster



Setup our GKE Cluster

- Back in your GCP account, launch a Cloudshell session
- Navigate to the directory **2-CreateCluster**
- Execute the script **setupenv.sh** and confirm that your credentials are correct



This step will take about 10 minutes to complete

Setup our GKE Cluster

- The **setupenv.sh** script will do the following:
  - Create a new GKE cluster
  - Deploy the Dynatrace OneAgent Operator to the cluster
  - Deploy the SockShop application to our cluster
    - Including creating the required Namespaces, storage objects and LoadBalancers

Validate our Cluster setup

- After about 10 minutes since running the setupenv.sh script, let's verify that everything is up and running
- Let's check and make sure our Deployments are Ready, run the following command in Cloudshell:
  - **`kubectl get deployments -n dev`**
- **Note:** This can also be validated in the GKE Console by selecting "Workloads" from the menu on the left side of the screen

```
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/2-CreateCluster (wms-keptn)$ kubectl get deployments -n dev
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
carts          1         1         1            1           90m
carts-db       1         1         1            1           90m
catalogue      1         1         1            1           90m
catalogue-db   1         1         1            1           90m
front-end      1         1         1            1           90m
orders         1         1         1            1           90m
orders-db      1         1         1            1           90m
payment        1         1         1            1           90m
queue-master   1         1         1            1           90m
rabbitmq       1         1         1            1           90m
shipping       1         1         1            1           90m
user           1         1         1            1           90m
user-db        1         1         1            1           90m
```

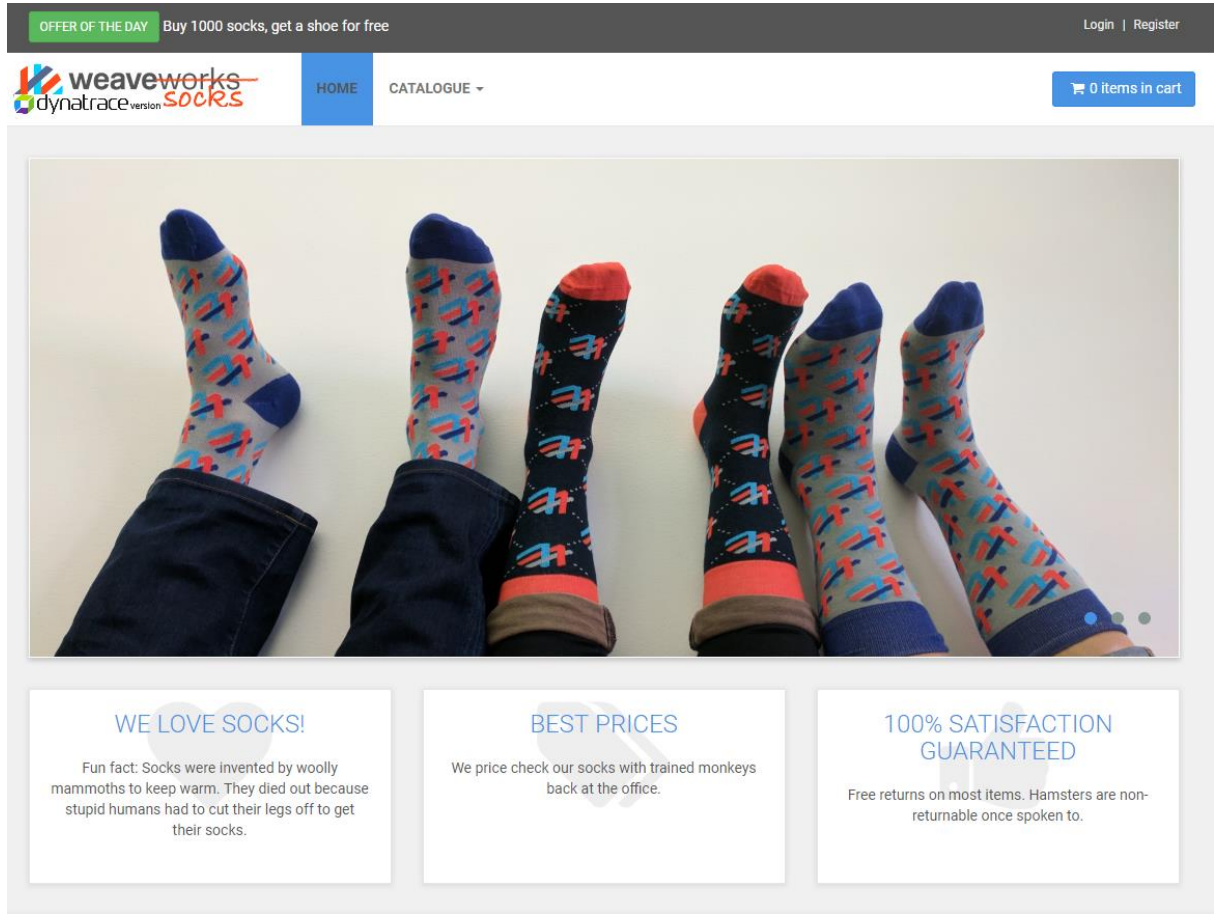
Validate our Cluster setup

- Lets ensure that the SockShop application services are also created by running the following command:
  - **`$kubectl get svc -n dev -o wide`**
- Our Carts and Front-end services should have External-IP addresses assigned
- **Note:** This can also be validated in the GKE Console by selecting "Services & Ingress" from the menu on the left side of the screen

```
wmsgclouddemouser1@cloudshell:~/dtacmworkshop/2-CreateCluster (wms-keptn)$ kubectl get svc -n dev -o wide
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)             AGE   SELECTOR
carts          LoadBalancer   10.31.250.178   35.232.184.170   80:30515/TCP        92m   app=carts
carts-db       ClusterIP      10.31.240.167   <none>           27017/TCP           92m   name=carts-db
catalogue      ClusterIP      10.31.242.33    <none>           80/TCP              92m   app=catalogue
catalogue-db   ClusterIP      10.31.248.14    <none>           3306/TCP            92m   name=catalogue-db
front-end      LoadBalancer   10.31.252.209   104.197.12.209   8080:30981/TCP      92m   app=front-end
orders         ClusterIP      10.31.255.165   <none>           80/TCP              92m   app=orders
orders-db      ClusterIP      10.31.255.198   <none>           27017/TCP           92m   name=orders-db
payment        ClusterIP      10.31.247.35    <none>           80/TCP              92m   app=payment
queue-master   ClusterIP      10.31.249.206   <none>           80/TCP              92m   app=queue-master
rabbitmq       ClusterIP      10.31.241.221   <none>           5672/TCP,9090/TCP   92m   name=rabbitmq
shipping       ClusterIP      10.31.246.77    <none>           80/TCP              92m   app=shipping
user           ClusterIP      10.31.252.24    <none>           80/TCP              92m   app=user
user-db        ClusterIP      10.31.241.83    <none>           27017/TCP           92m   io.kompose.service=user-db
```

Validate our Cluster setup

- Let's launch our application in a web browser by copying and pasting the External-IP and Port of the front-end service. In the previous example it would be: http://104.197.12.209:8080
- The SockShop homepage should appear in your browser



Clean up script placeholder

- In Terminal/Cloudshell execute the script: **cleanup.sh**

- This will remove the dev, production and istio-system namespaces (and their associated objects) so we have a clean cluster

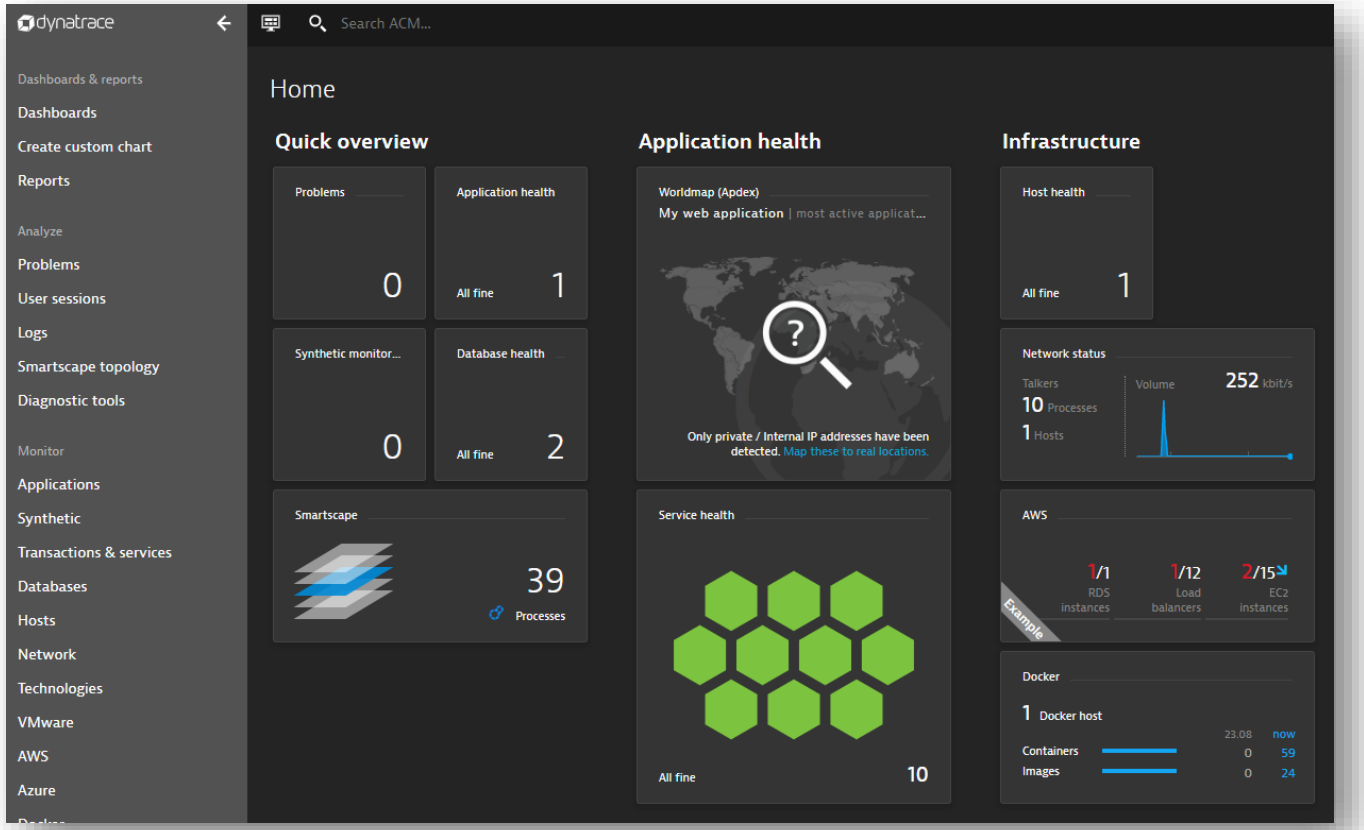
- After this completes, we will redeploy the sock shop application in the dev namespace. We only need to execute the script: **deploy-sockshop.sh** and then wait a few minutes

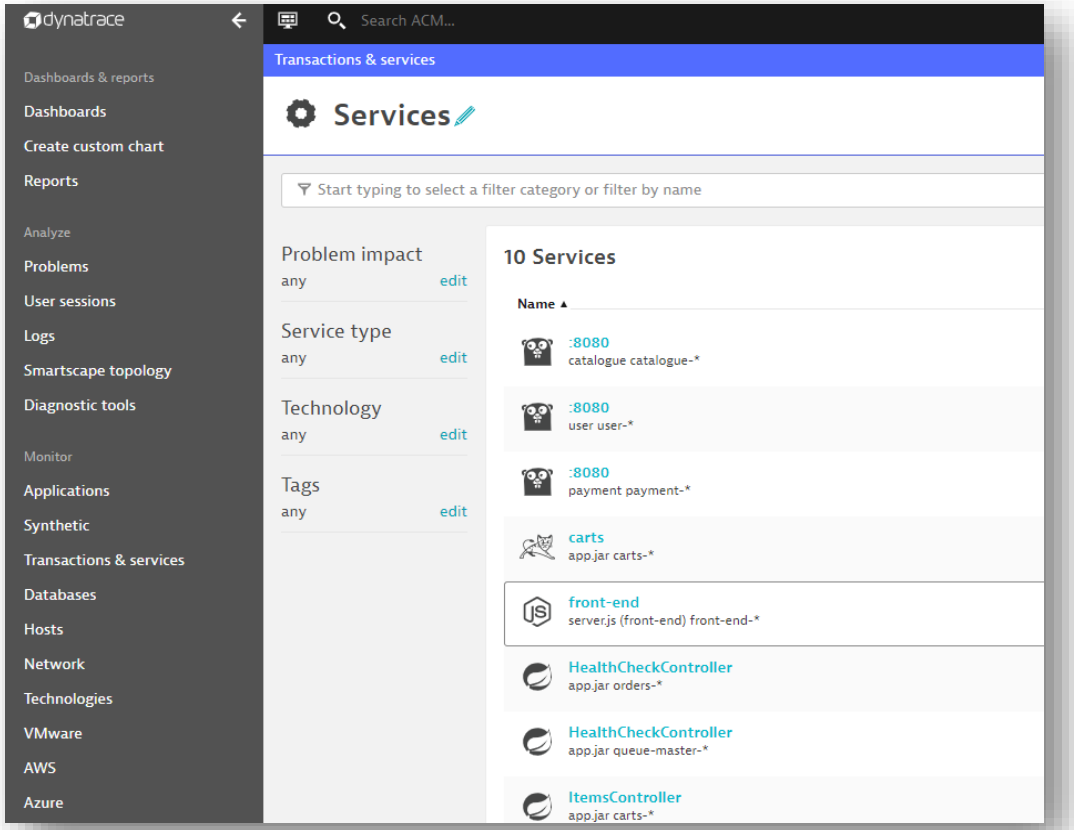
- Then check if the SockShop app is back up and running

```
$kubectl get svc -n dev -o wide
```

Validate Dynatrace

- Now that we know our application is deployed and running successfully, we can validate that everything is being displayed in Dynatrace
- Navigate back to Dynatrace and you should see data for Host, Process and Services





Deploying Jenkins

Deploy Jenkins

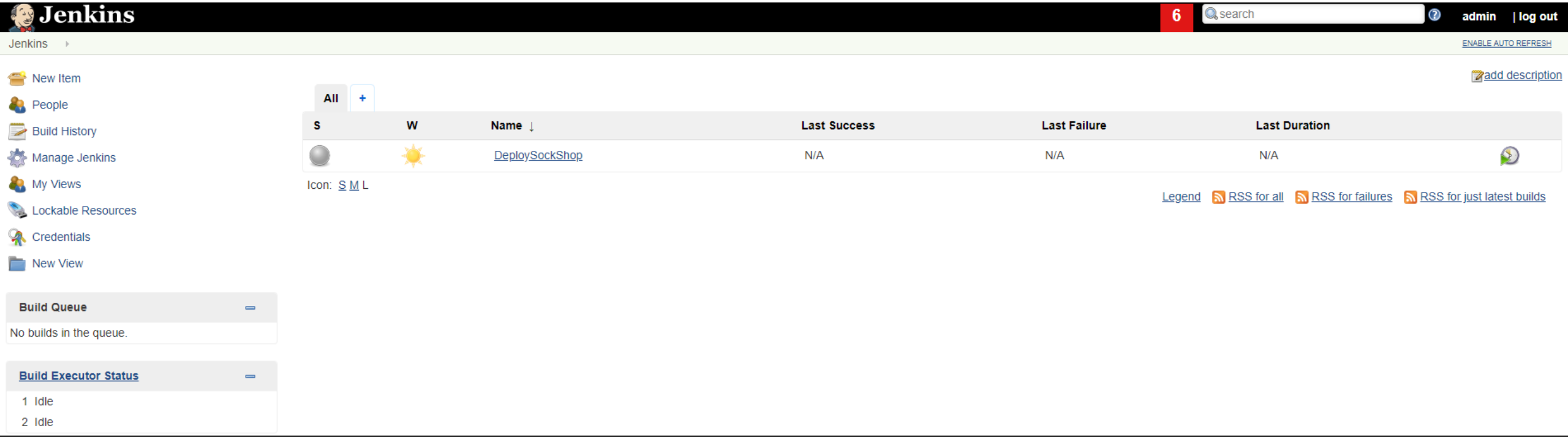
- In Terminal/Cloudshell execute the script: **deployJenkins.sh** in the directory **3-DeployJenkins**
- This will deploy and startup Jenkins, import our pipeline and output the credentials

```
wmsgclouddemouser1@cloudshell:~/dynatrace-sockshop (wms-keptn)$ ./deployJenkins.sh
namespace/cicd created
persistentvolumeclaim/jenkins-jobs created
persistentvolumeclaim/jenkins-workspaces created
persistentvolumeclaim/maven-repo created
service/jenkins created
deployment.extensions/jenkins-deployment created
clusterrolebinding.rbac.authorization.k8s.io/jenkins-rbac created
secret/jenkins-secret created
Waiting for Jenkins to start...
-------------------------------------------------
Jenkins is running at : http://35.226.212.58:24711
Username is : admin
Password is : AiTx4u8VyUV8tCKk
```

Note: You should change the default password!

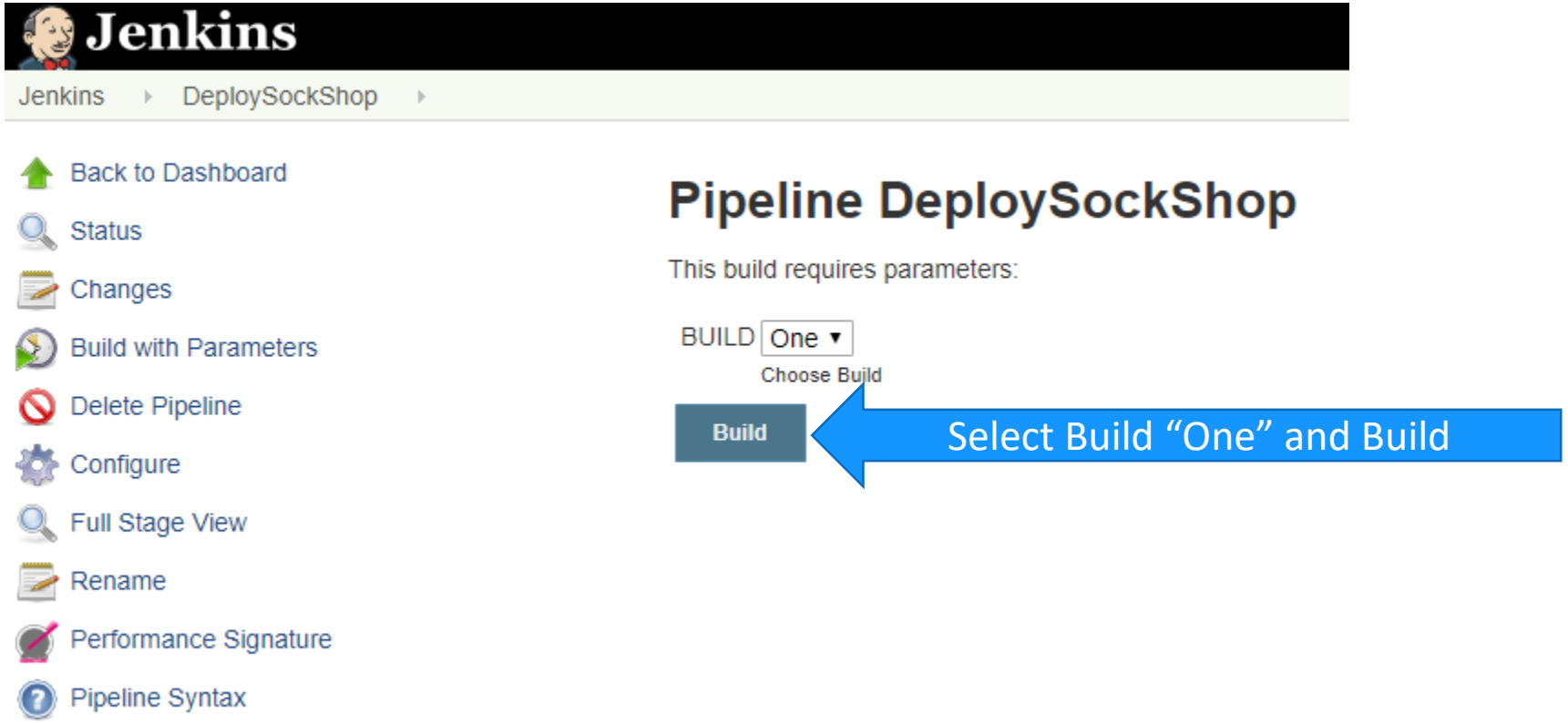
Jenkins Setup

- After Jenkins has successfully deployed, navigate to the Jenkins URL and login with the previously provided username and password (again, recommend you change the password ☺



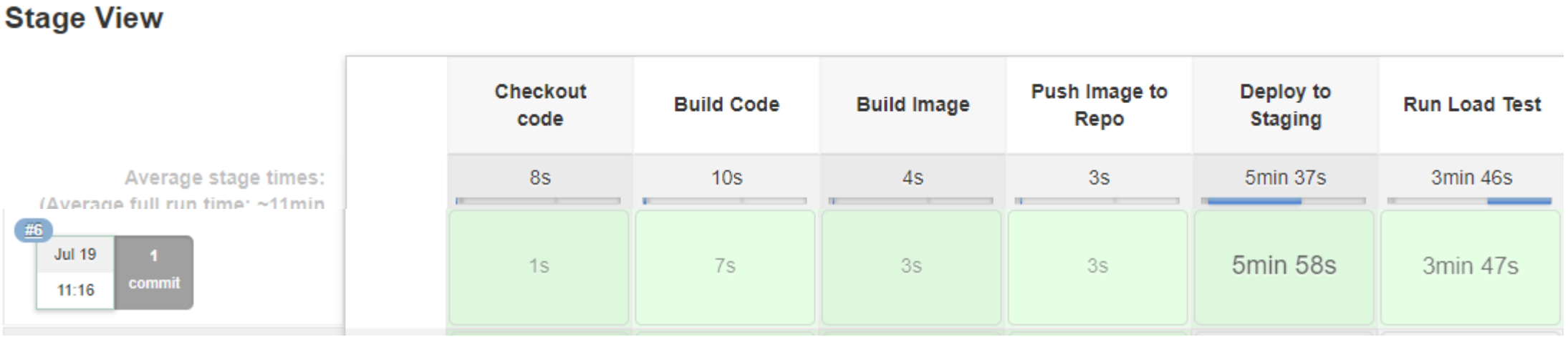
Run our first pipeline

- On the Jenkins main screen we should see the “DeployShockShop” pipeline, select the pipeline by clicking it



If all goes well...

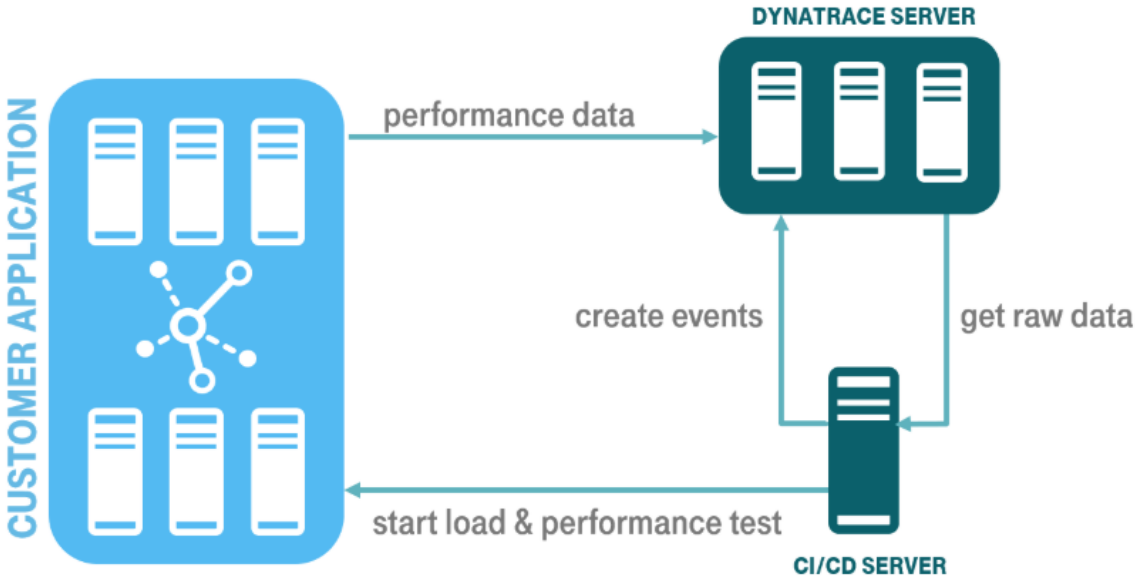
- The pipeline should run successfully (Note: this will take about 10 min)



Dynatrace Integration

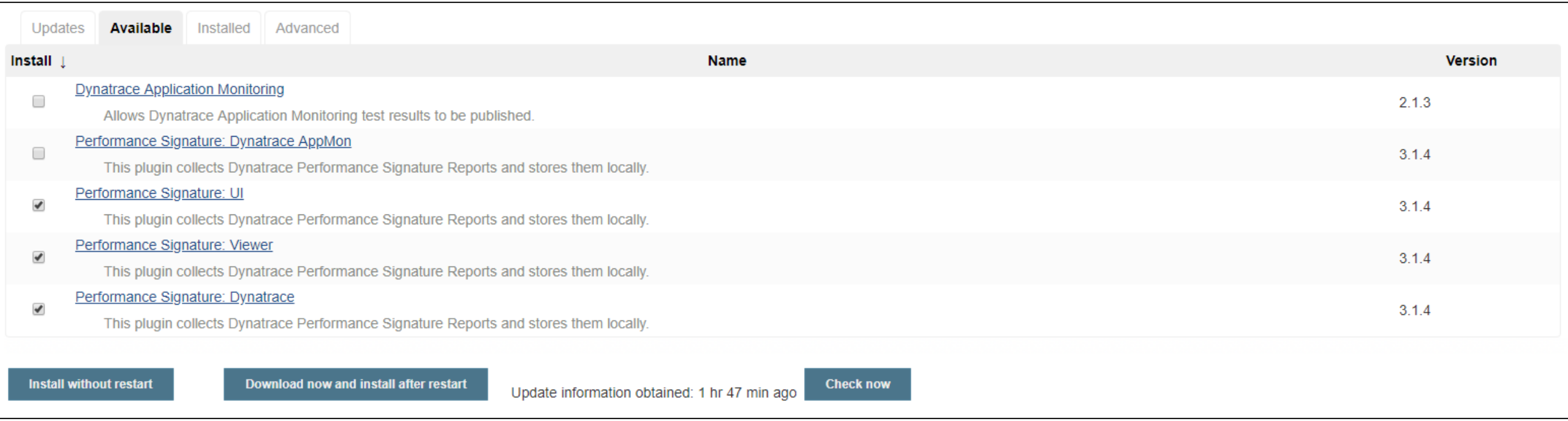
Dynatrace Integration

- GitHub Repo and Example: https://github.com/jenkinsci/performance-signature-dynatrace-plugin/blob/master/README.md
- Integration via the Dynatrace APIs (Timeseries and Events)
- Based on a Performance Signature of a release (monspec)



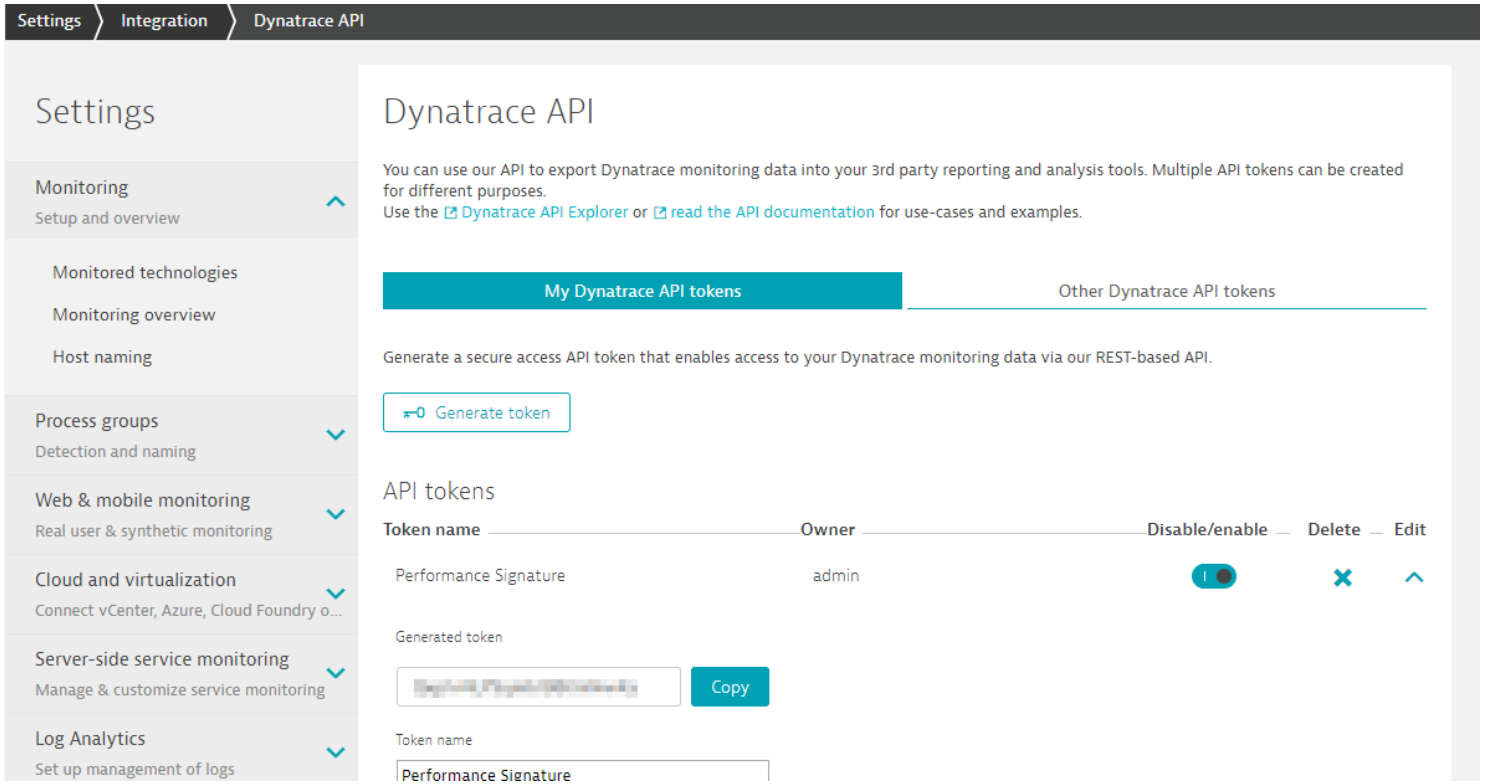
Install Jenkins Plugin

- Navigate to **Manage Jenkins** -> **Manage Plugins** page and switch to the Available tab. Search for the "Dynatrace" keyword, select the 3 options below and click “Install without restart”.



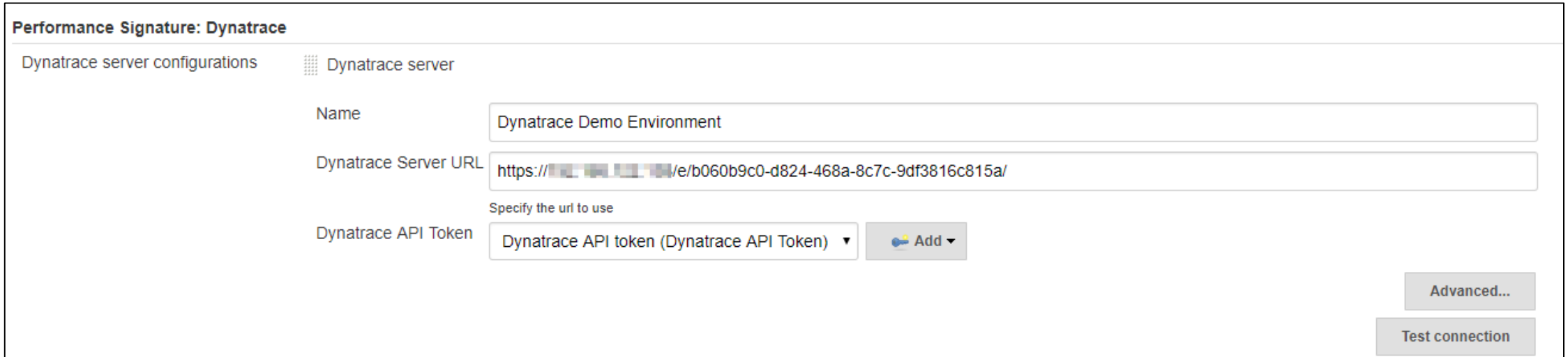
Dynatrace API

- A Dynatrace API Token is needed for the Integration (Settings -> Integration -> Dynatrace API)
  - Only need the default top 3 permissions (This can be the same API Token as created in earlier steps)



Jenkins Configuration

- Navigate to Manage Jenkins -> Configure System -> Performance Signature: Dynatrace SaaS/Managed
- Enter name of the
  - Server (ex. Dynatrace Server)
  - Your Tenant URL (https://xxxxxxxx.live.dynatrace.com) – SaaS, (https://asdfa.Dynatrace-managed.com/e/asdfafa) - Managed
  - API Token
  - Note: You many need to Save and exit this screen and come back for the "Add" dropdown to works, thanks to a bug in Jenkins
- Test the Connection and Save



Adding Events to our Pipeline

- There are three main events we will be adding to our Deployment Pipeline:
  - createDynatraceDeploymentEvent – Sends the Deployment Event to Dynatrace
  - recordDynatrace Session – Sets start/end time for when Perf Test beings
  - perfSigDynatraceReports – Queries all the metrics during the perf test timeframe and compares against the spec file. If metrics violate the spec file, the pipeline stage is failed

Adding Events to our Pipeline

- In the DeploySockShop pipeline, click “Configure” and scroll down to the Pipeline script

Adding Events to our Pipeline – Deployment Events

- First we want to add a Deployment Event to our Pipeline so every time we deploy our build to staging, Dynatrace will know about it

```
stage('Deploy to Staging'){
            steps{
                container('kubectl'){
                    echo 'Deployment canary build...'
                    createDynatraceDeploymentEvent(envId: 'Dynatrace Server',
                        tagMatchRules: [[meTypes: [[meType: 'SERVICE']],
                        tags: [[context: 'KUBERNETES', key: 'app', value: 'carts'],
                                [context: 'KUBERNETES', key: 'stage', value: 'dev']]]]) {
                    // some block
                    }
                    script{
                        if (params.BUILD == "One") {
                            sh 'ls $WORKSPACE'
                            sh 'kubectl apply -f $WORKSPACE/manifests/sockshop-app/dev/carts2.yml'
                            echo "Waiting for carts service to start..."
                            sleep 350
                        } else {
                            sh 'ls $WORKSPACE'
                            sh 'kubectl apply -f $WORKSPACE/manifests/sockshop-app/canary/carts2-canary.yml'
                            echo "Waiting for carts service to start..."
                            sleep 350
                        }
                    }
                }
            }
        }
```

Adding Events to our Pipeline – Record Perf Test

- Next we want to wrap our Loadtest stage with the recordDynatraceSession function, so we know when tests start and stop

```
stage ('Run Load Test'){
            steps{
                container('kubectl'){
                    echo 'Get Carts URL'
                    script{
                        dir("loadtest"){
                            sh "chmod +x cartstest.sh"
                            sh "./cartstest.sh"

                        }
                    }
                }
                container('jmeter'){
                    script{
                        recordDynatraceSession(envId: 'Dynatrace Server',
                                tagMatchRules: [[meTypes: [[meType: 'SERVICE']],
                                tags: [[context: 'KUBERNETES', key: 'app', value: 'carts'],
                                        [context: 'KUBERNETES', key: 'stage', value: 'dev']]]],
                                testCase: 'loadtest') {

                        if (params.BUILD == "One"){
                            dir("loadtest"){
                                sh "jmeter -n -t carts_load1.jmx -l testresults.jtl"
                            }
                        } else {
                            dir("loadtest"){
                                sh "jmeter -n -t carts_load2.jmx -l testresults.jtl"
                            }
                        }

                        }
                    }
                }
            }
        }
```

Adding Events to our Pipeline – Compare results to monspec

- After the load test is completed, we want to validate the test against our monspec file (see next slide), to determine if our build meets quality or not
- Add the below stage (“Dynatrace Quality Gate”) to the very end of our pipeline

```
stage ('Dynatrace Quality Gate'){
            steps{
              perfSigDynatraceReports envId: 'Dynatrace Server', nonFunctionalFailure: 2, specFile: 'monspec.json'
            }
        }
```

Sample Monspec File

```
{
    "spec_version": "2.0",
    "timeseries": [{
            "timeseriesId": "com.dynatrace.builtin:service.responsetime",
            "aggregation": "avg",
            "tags": “[Kubernetes]app:carts,[Kubernetes]stage:dev",
            "upperWarning": 1000,
            "upperSevere": 3000
        }
    ]
}
```

Adding Events to our Pipeline

- Once the events are added to our Pipeline we can apply and save

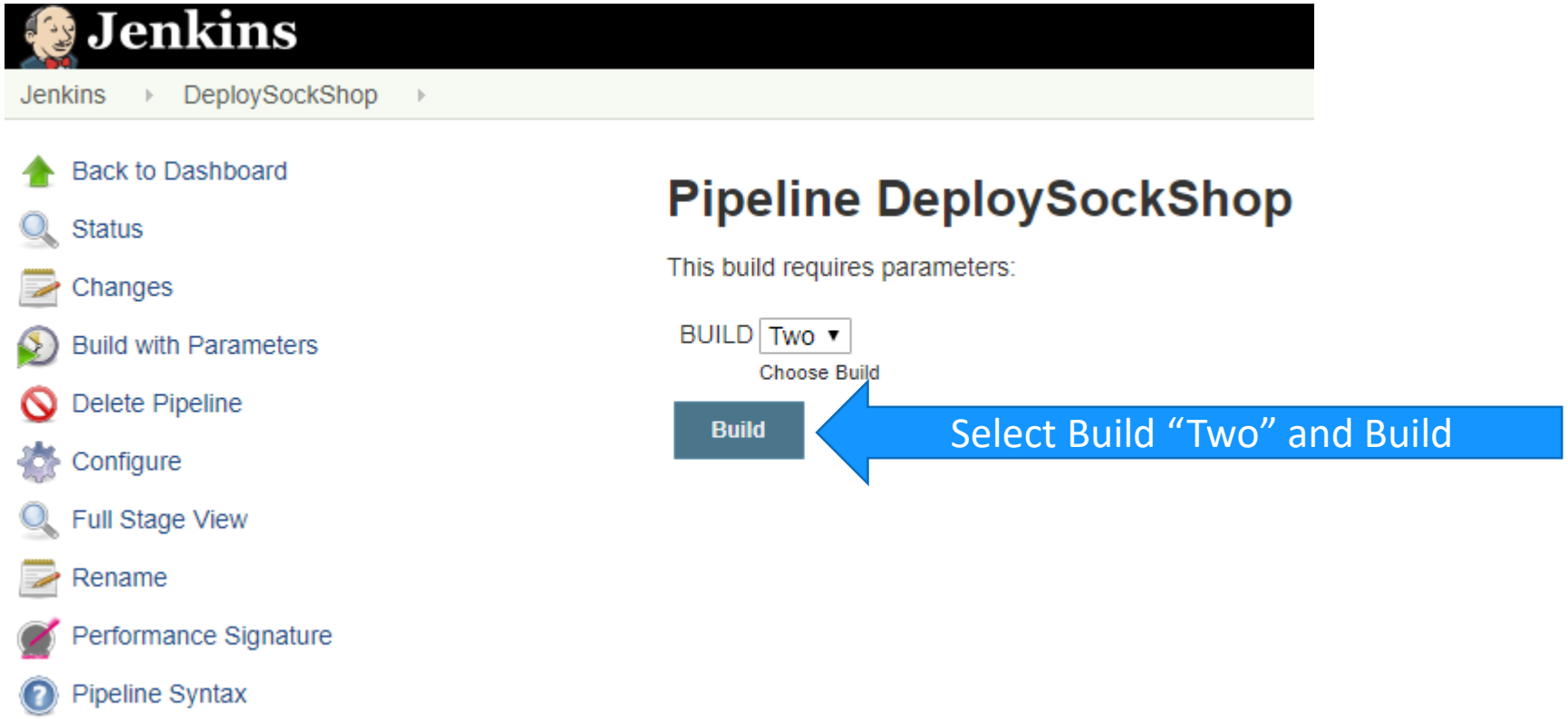
- Note: There is a sample Pipeline with the events already added in the root of the repository titled "PipelineWithIntegration.txt"

- You can use this as a reference when editing your pipeline
  - Note: You will need to make sure you change the name of the Dynatrace Environment to the same thing as you named your connection earlier. This file uses the name "Dynatrace Server" as the Dynatrace Environment Name.

Let’s test our Integration

Deploy Build Two

- After updating and saving our Pipeline to include the Dynatrace Performance Events, let's re-run our pipeline with build Two



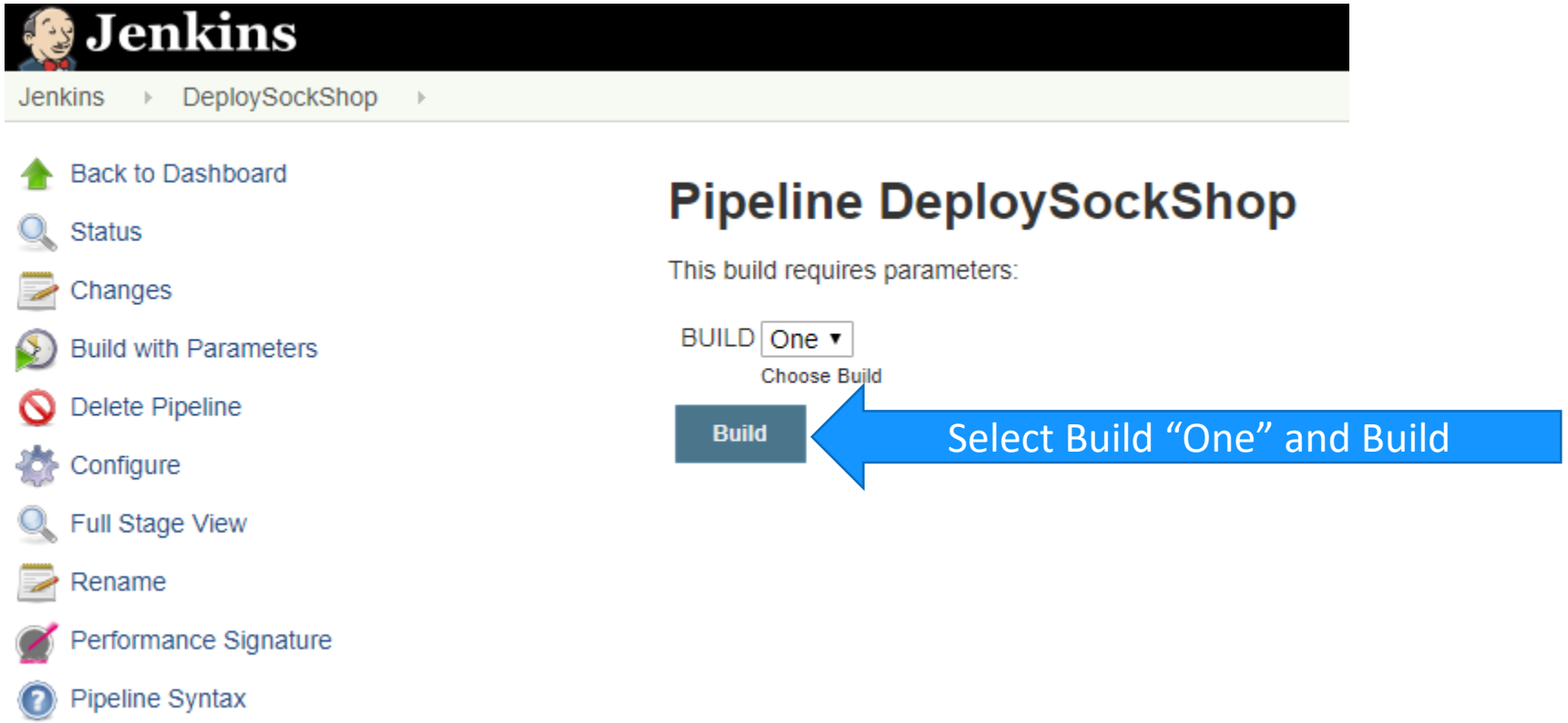
Now our build should fail

Stage View

	Checkout code	Build Code	Build Image	Push Image to Repo	Deploy to Staging	Run Load Test	Dynatrace Quality Gate
Average stage times: (Average full run time: ~11min 29s)	8s	10s	4s	3s	5min 37s	3min 46s	1s
#7 Jul 19 11:28 — No Changes	679ms	6s	3s	3s	5min 58s	5min 21s	1s failed
#6 Jul 19 11:16 — 1 commit	1s	7s	3s	3s	5min 58s	3min 47s	1s

Deploy Ansible Tower

- If we redeploy using Build #1, this build should pass our Dynatrace Quality Gate



Auto-remediation with Ansible Tower

Ansible and Ansible Tower

- Runbook – definition: a compilation of routine procedures and operations that the system operator (administrator) carries out
- Ansible
  - Simple automation language + automation engine

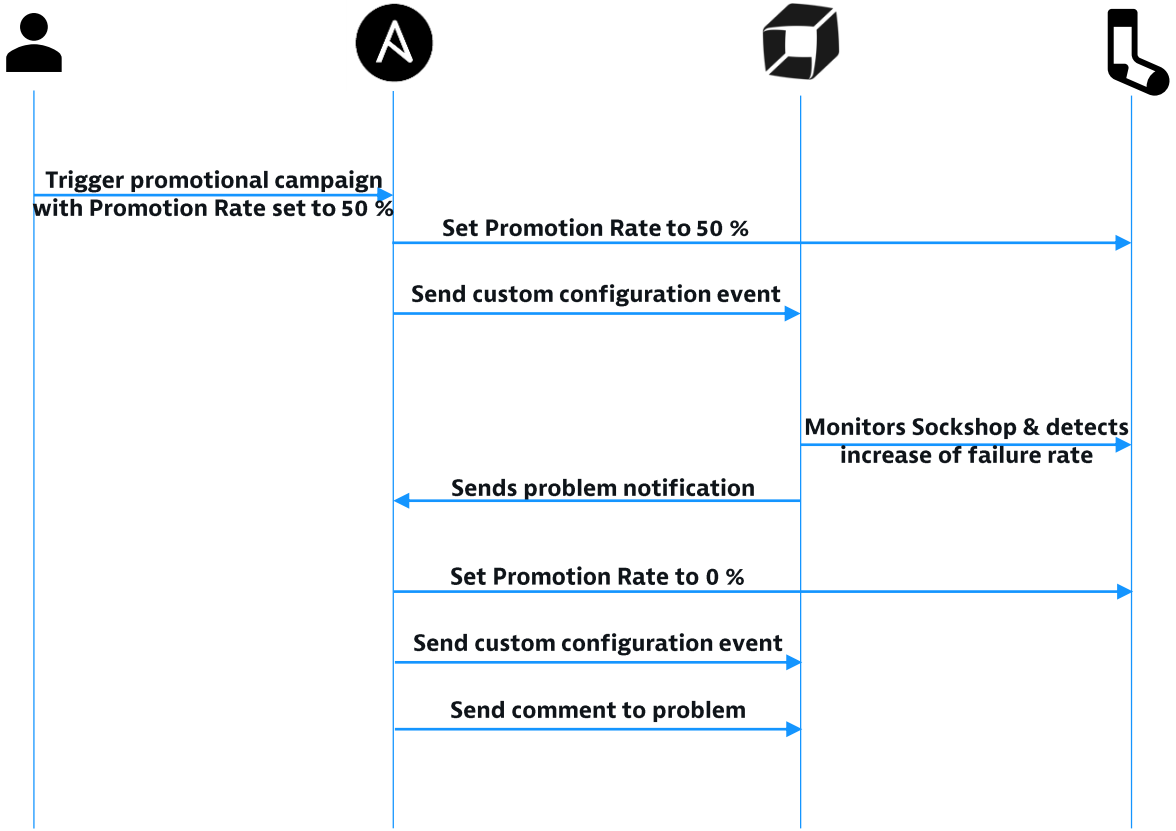
  - Human readable (yaml)
- Ansible Tower
  - Web UI
  - Management of runbooks
  - Powerful API for automation

RED HAT® ANSIBLE® Tower

Promotional campaign

- We'll be implementing a promotional campaign for our socks
- This promotion will be randomly available to customers
- The carts service has an endpoint allowing us to provide the percentage of users that will benefit from the promotional gift
- The problem is that the promotion implementation is defective ☹
- Nobody knows about it yet and this could potentially have disastrous effect on the brand and business reputation

Runbook automation workflow



Deploy Ansible Tower

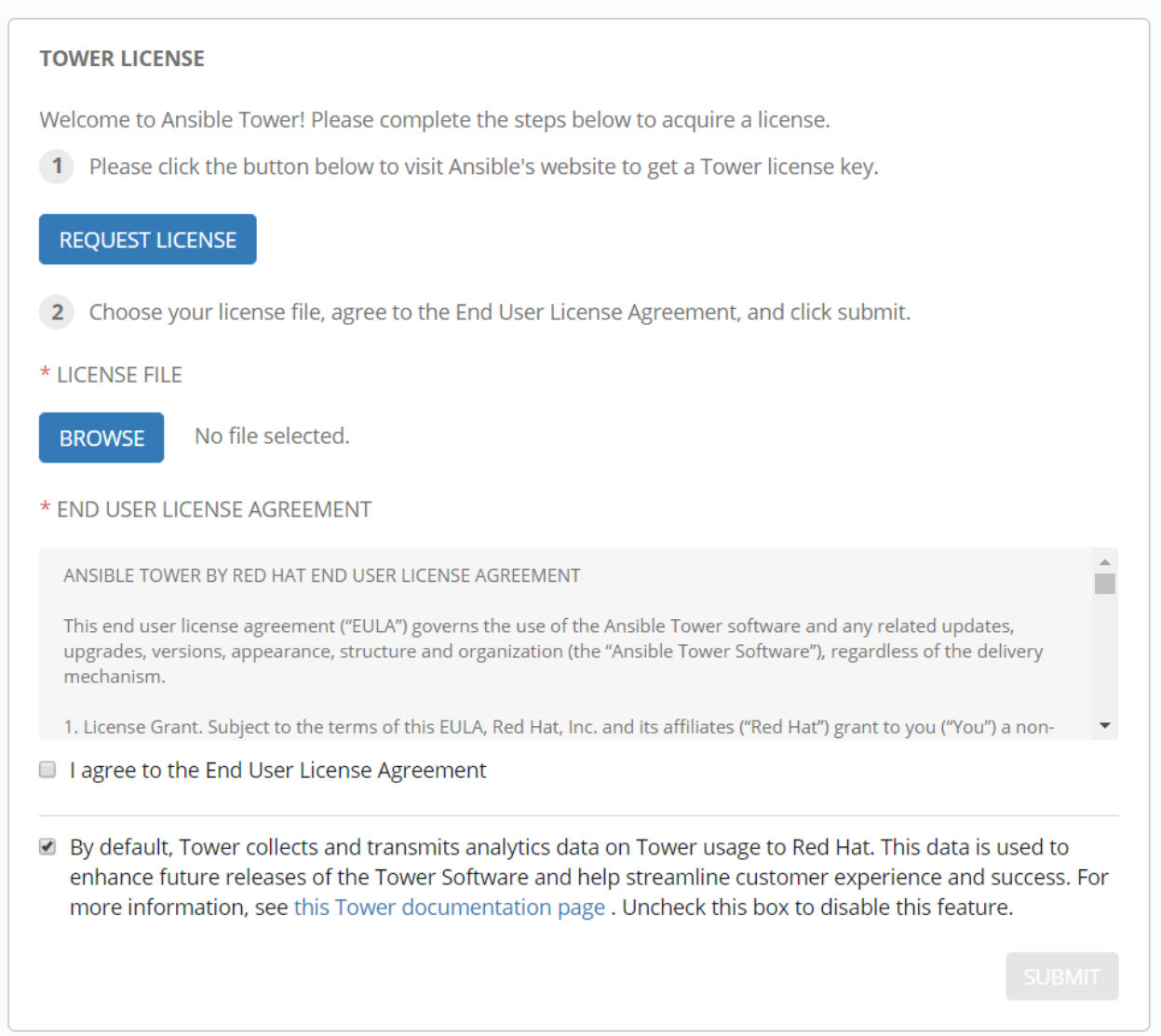
- In Terminal/Cloudshell execute the script: **deployTower.sh** in the directory **4-DeployTower**
- This will deploy and startup Ansible Tower. This will also configure Tower, import projects, inventories, credentials (DT API token) and playbook templates.
- Take a note of the Ansible Job URL displayed by the script. You will need it for a next step.

```
                                 Dload  Upload   Total   Spent    Left  Speed
100  3059  100  2896  100   163   4009    225 --:--:-- --:--:-- --:--:--  4005
REMEDIATION_TEMPLATE_ID:  35
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3357  100  3053  100   304   3659    364 --:--:-- --:--:-- --:--:--  3656
STOP_CAMPAIGN_ID:  36
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3395  100  3054  100   341   4553    508 --:--:-- --:--:-- --:--:--  4558
START_CAMPAIGN_ID:  37
-------------------------------------------------
Ansible has been configured successfully! Copy the following URL to set it as an Ansible Job URL in the Dynatrace notification settings:
https://34.69.184.20/#/templates/job_template/35
-------------------------------------------------
```

Log in the Ansible Tower console

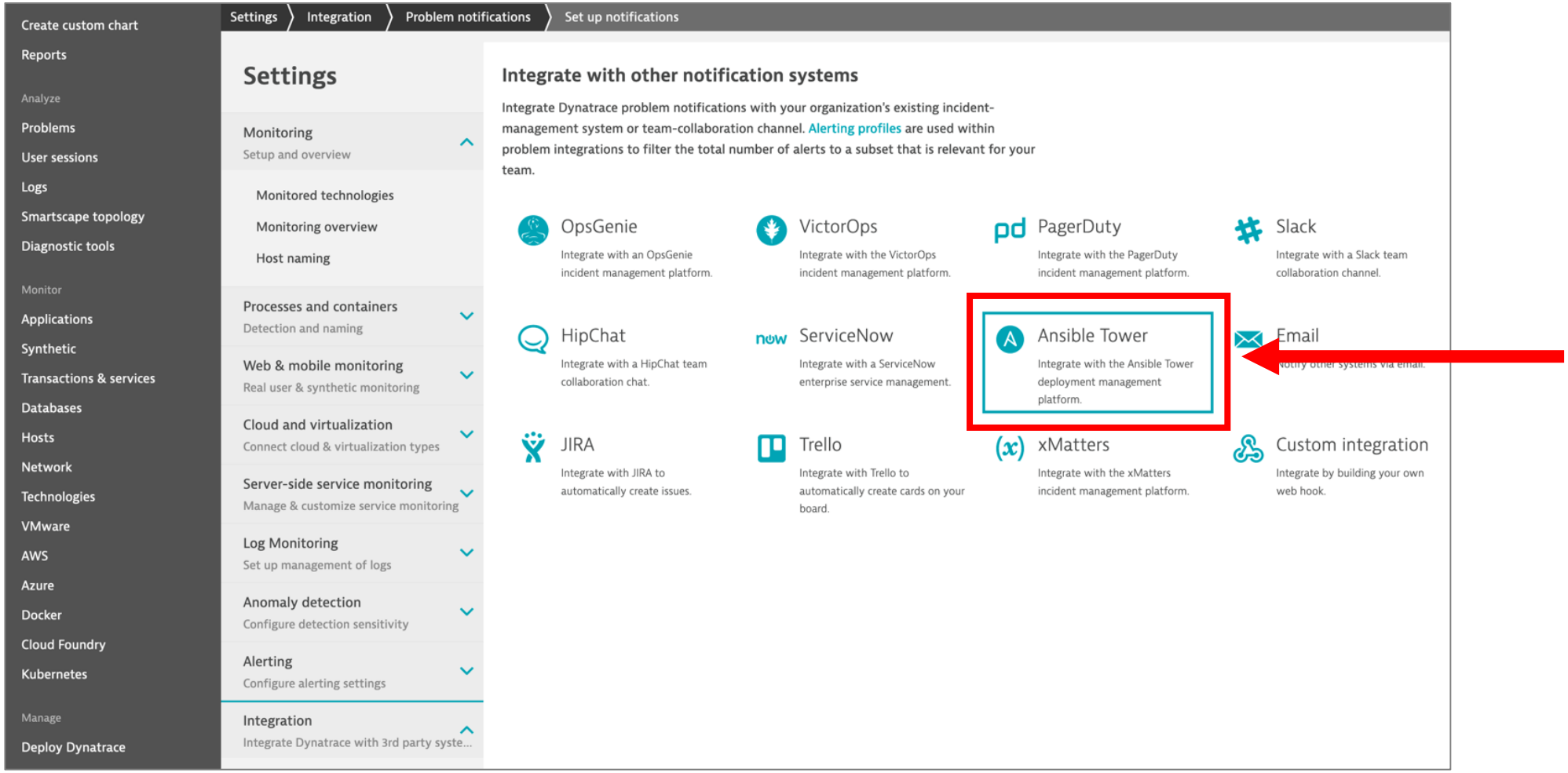
- Use the job template URL you copied earlier, without the path, and copy it in your browser to load the Ansible Tower console
  - For example, if the job template URL was : https://34.69.184.20/#/templates/job_template/35

  then the console URL is : https://34.69.184.20
- You will be prompted to provide a license file and accept the license agreement
  - If you don't already have your license file, click the button to request a free trial license from Red Hat (delivered via email)
- Log in Ansible Tower
  - username : admin
  - password : dynatrace



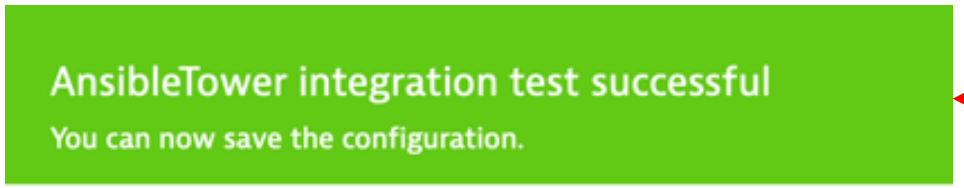
Configure Dynatrace Problem Notification

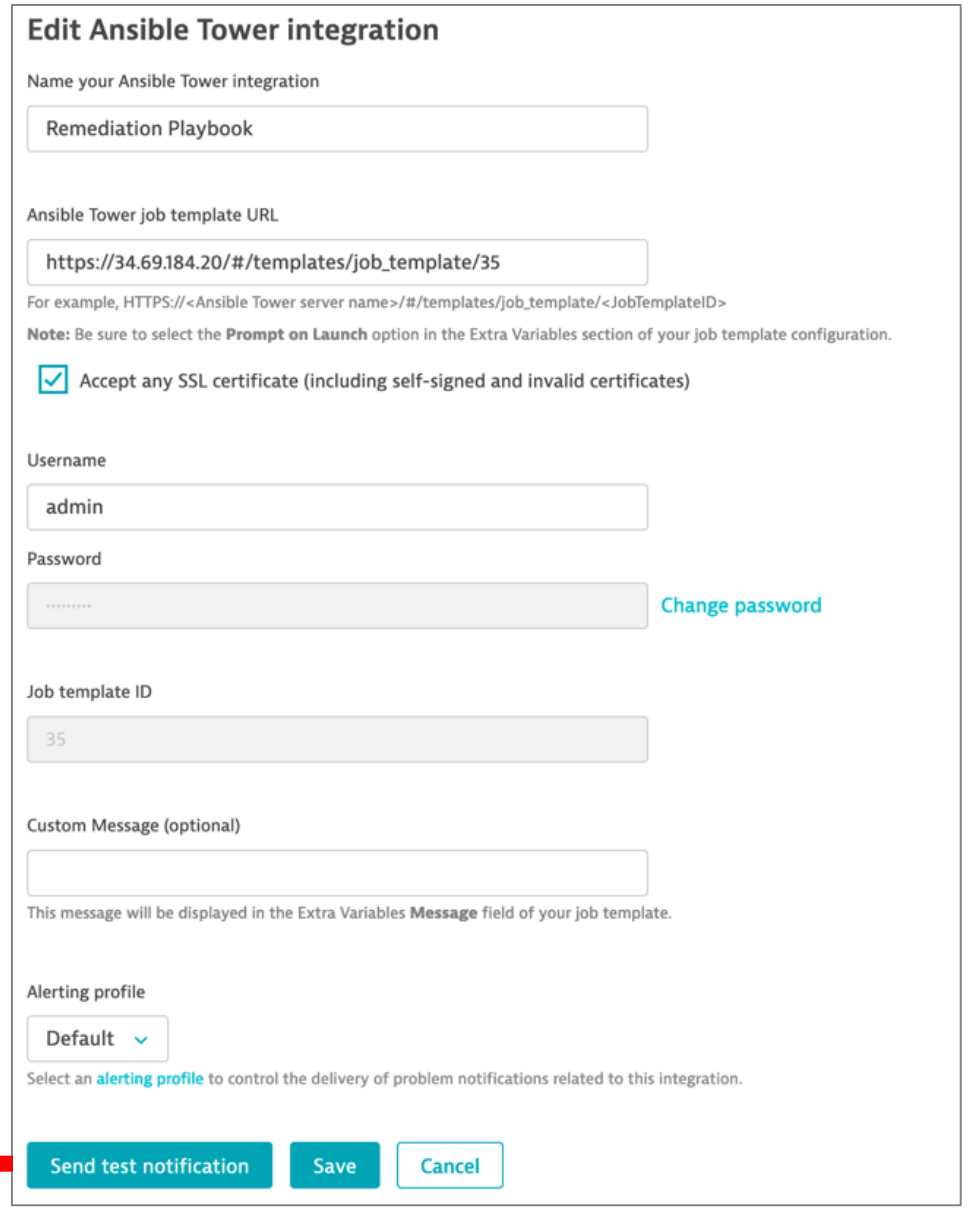
- In the Dynatrace UI, navigate to Settings -> Integration -> Problem Notification -> Ansible Tower



Configure Ansible Tower integration

- Enter the Ansible Tower job template URL you were provided after the deployment
- Enter the credentials to access Ansible
  - username : admin
  - password : Dynatrace
- Click the Sent test notification button to validate your configuration
  - on success, a green confirmation message will be displayed
- Save your configuration

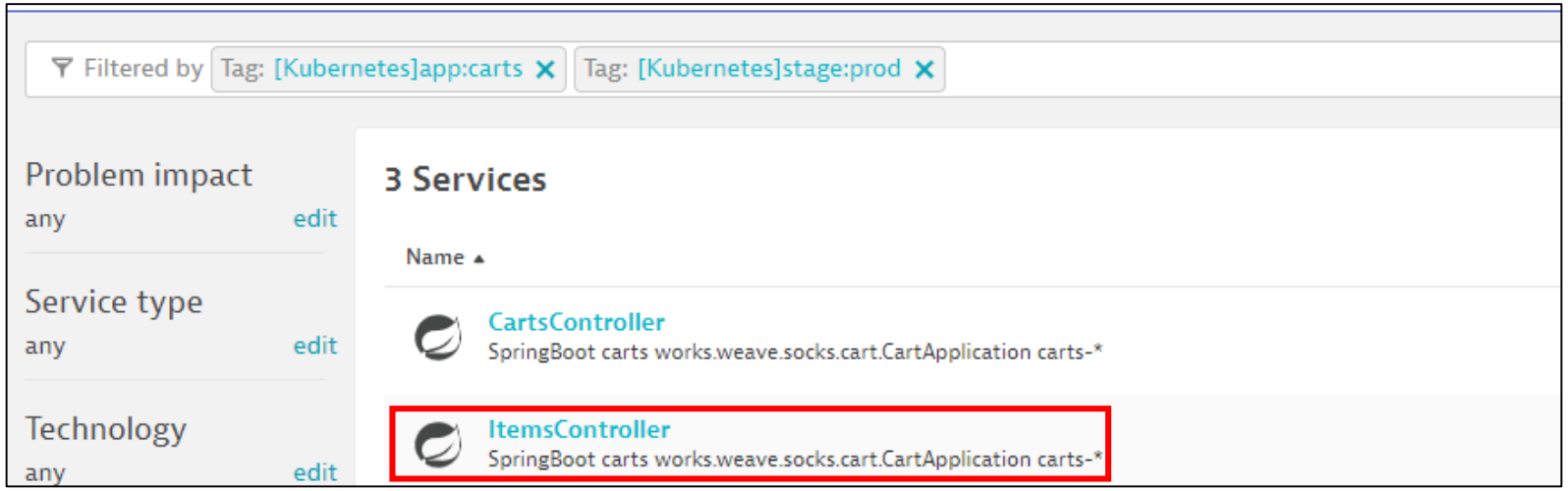




Adjust Anomaly Detection

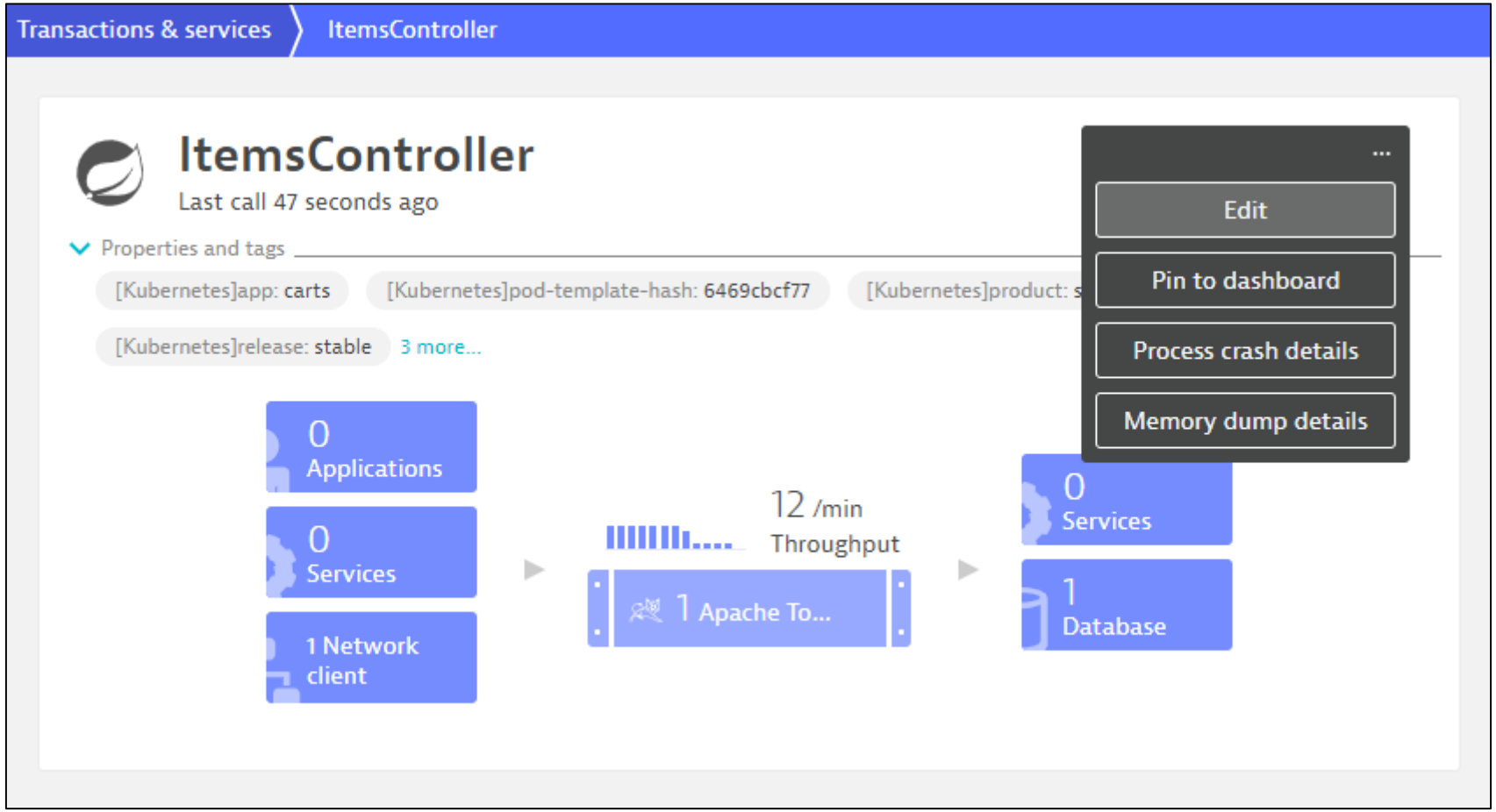
Both problem and anomaly detection in Dynatrace leverage AI technology. This means that the AI learns how each and every microservice behaves and baselines them. Therefore, in a demo scenario like we have right now, we have to override the AI engine with user-defined values to allow the creation of problems due to an artificial increase of a failure rate. (**Please note:** if we would have the application running and simulate end-user traffic for a couple of hours/days there would be no need for this step.)

In your Dynatrace tenant, navigate to "Transaction & services" and filter by:
[Kubernetes]app:carts and *[Kubernetes]environment:production*



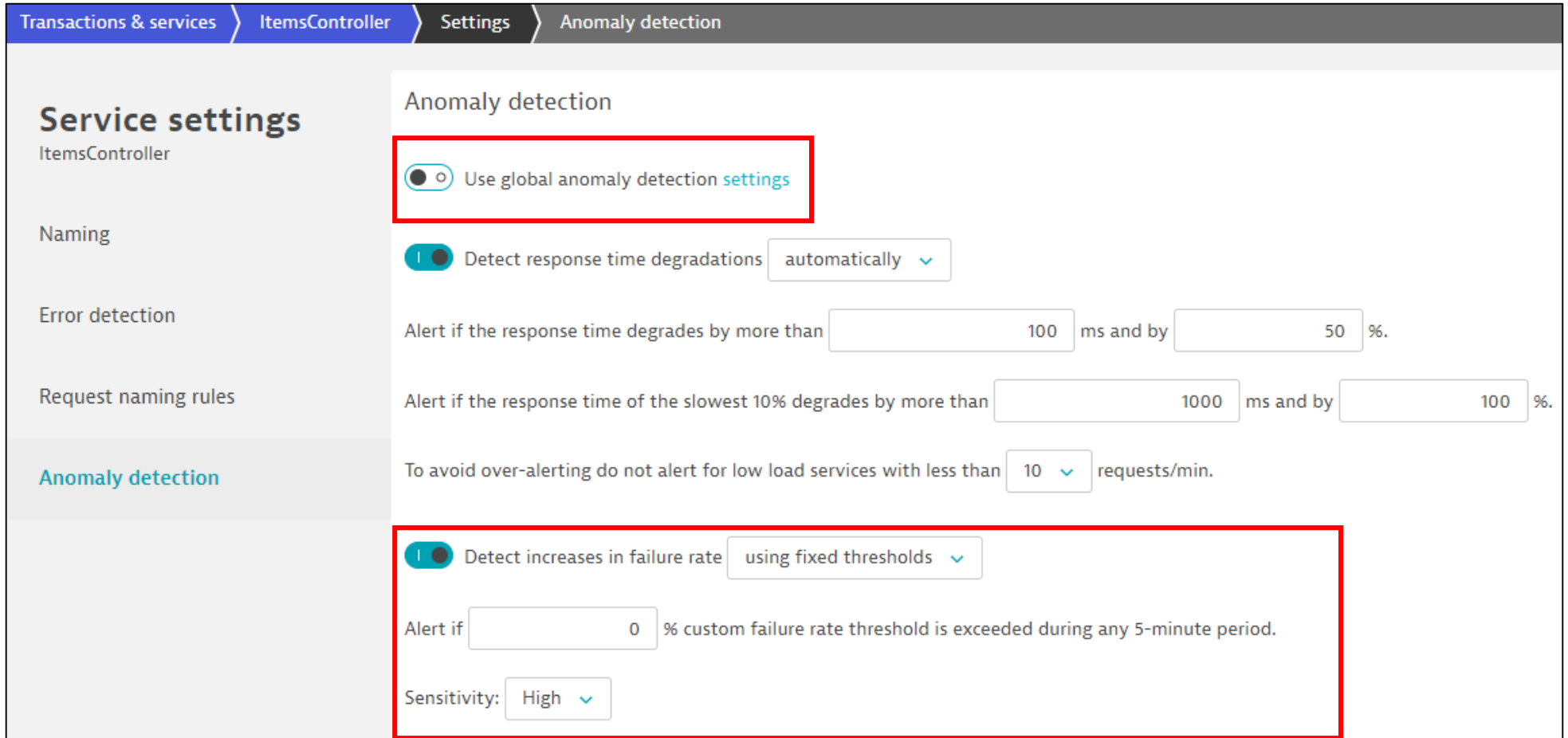
Adjust Anomaly Detection (Cont.)

- Click on the **ItemsController** and then on the three dots (...) next to the service name. Click on *Edit*



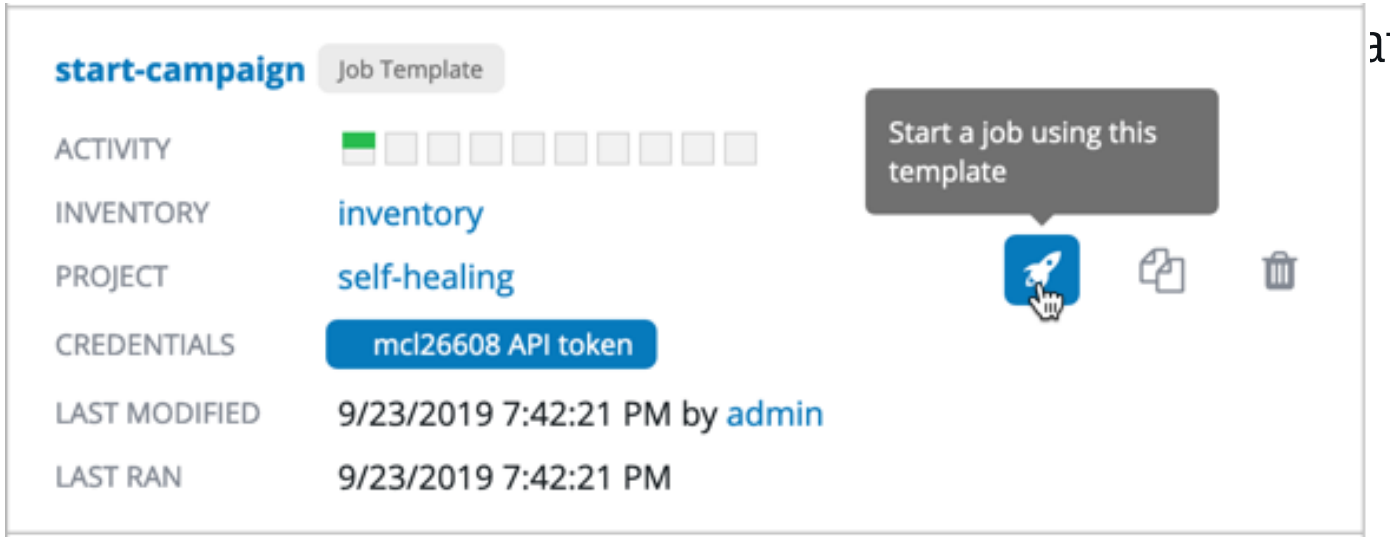
Adjust Anomaly Detection (Cont.)

On the next screen, edit the anomaly detection settings as seen in the following screenshot. - **Global anomaly detection** has to be **turned off** - Detect increases in **failure rate** using **fixed thresholds** - Alert if **0 %** custom failure rate threshold is exceed during any 5-minute period. - Sensitivity: **High**



Launch Campaign playbook

- Navigate back to the Ansible Tower UI
- From the side menu, navigate to Resources -> Templates
- Click on the rocket icon to launch the start-campaign playbook
- Hit Next on the prompt popup window and then Launch

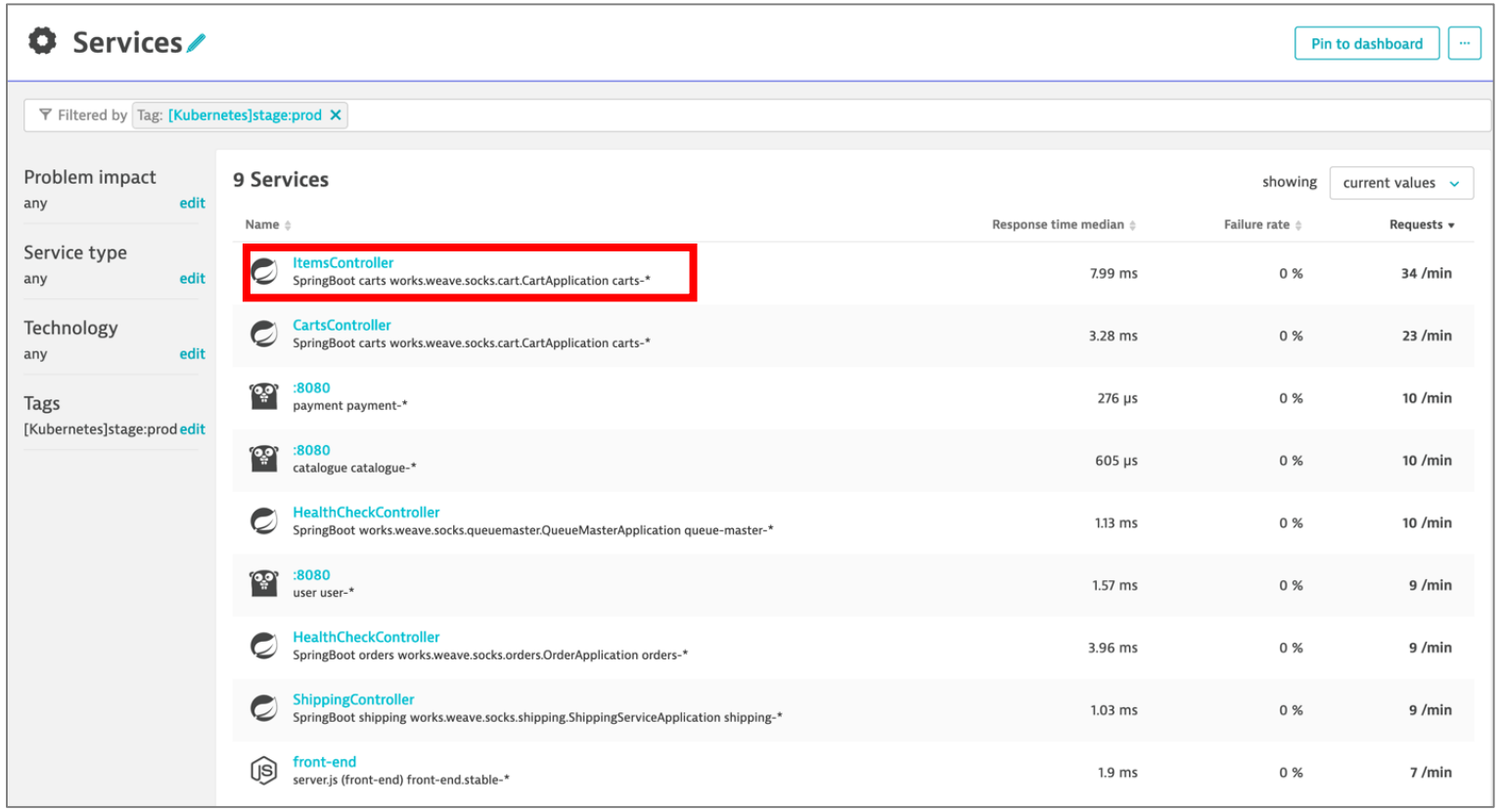


ate that the recap shows OK=2

```
5  PLAY [localhost] ***************************************************************
6
7  TASK [update percentage of promotion for carts service] ************************
8  ok: [localhost]
9
10 TASK [update promotion configuration for carts service] ************************
11 ok: [localhost]
12
13 PLAY RECAP *********************************************************************
14 localhost                  : ok=2    changed=0    unreachable=0    failed=0
15
```

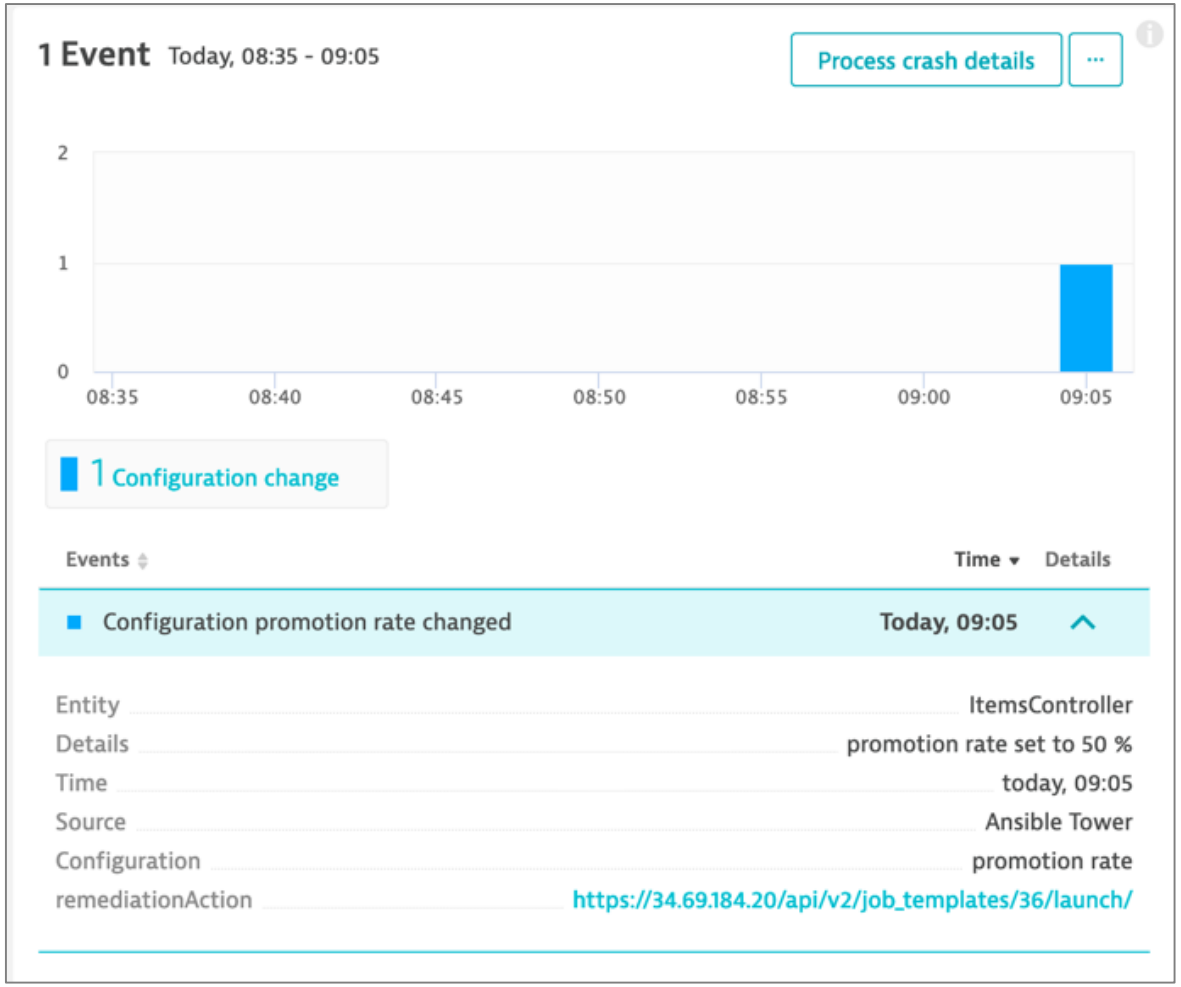
Observe the promotional campaign

- In the Dynatrace console, navigate to Transactions & Services and drill-down into the production ItemsController service
  - Tip : filter by tag : [Kubernetes]stage:prod to only display services in production



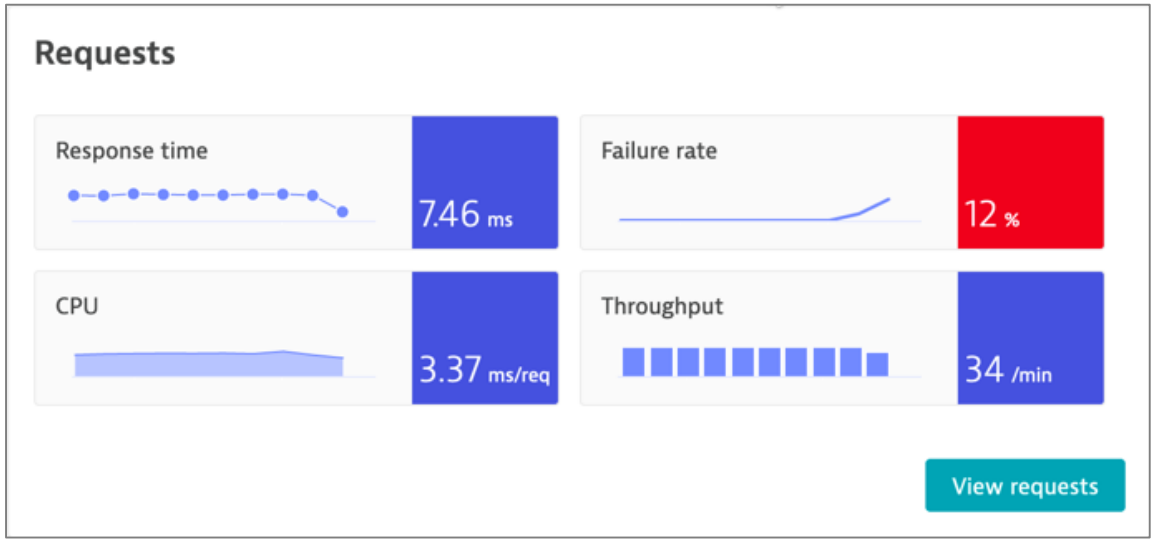
Observe the promotional campaign

- Look at the Service events.
- The Start Campaign playbook have notified Dynatrace when the promotional rate was changed to 50%



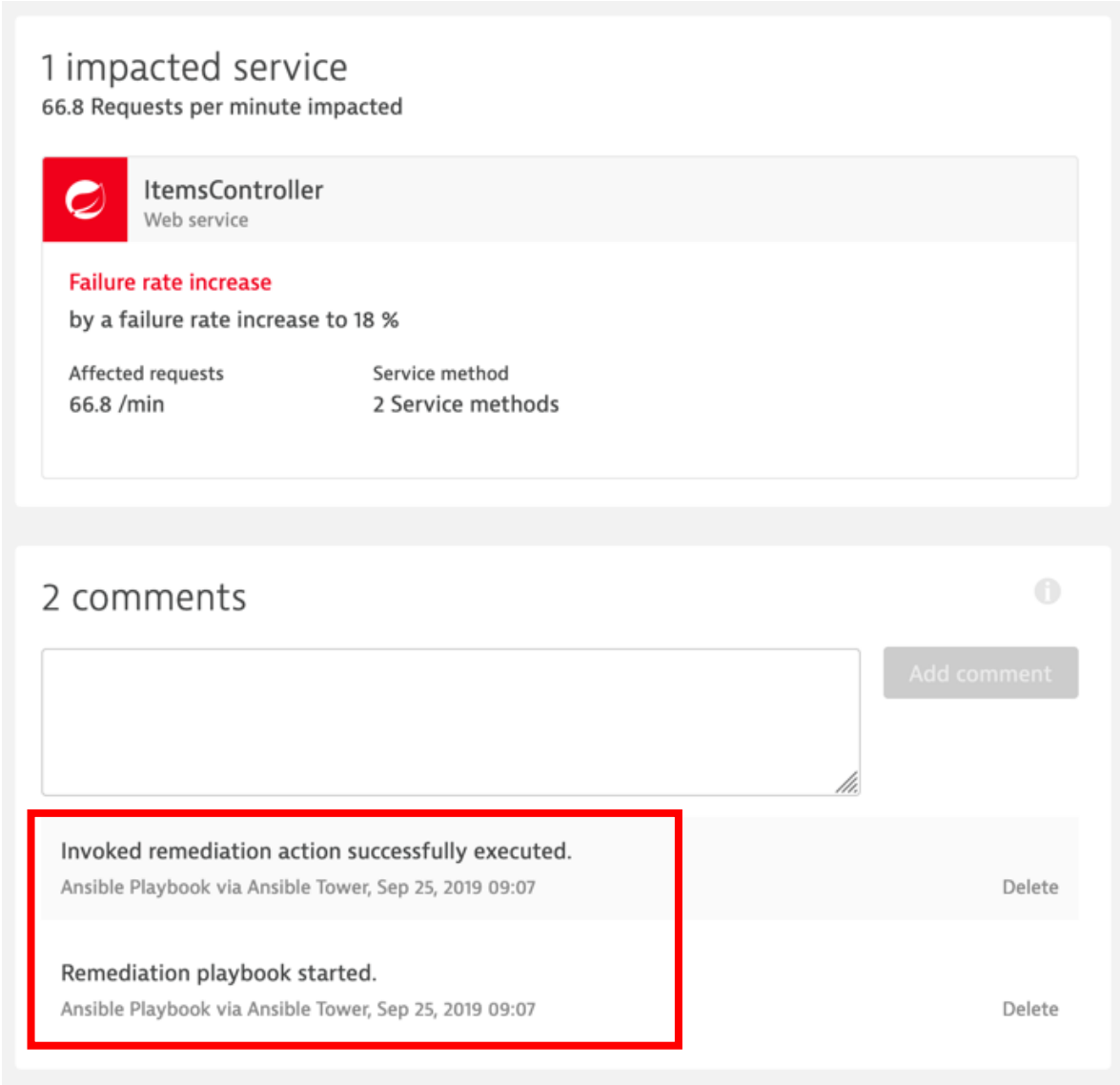
Observe the promotional campaign

- You should see the Failure Rate increasing, eventually leading to Dynatrace detecting a Problem
  - You might need to refresh your browser a few times



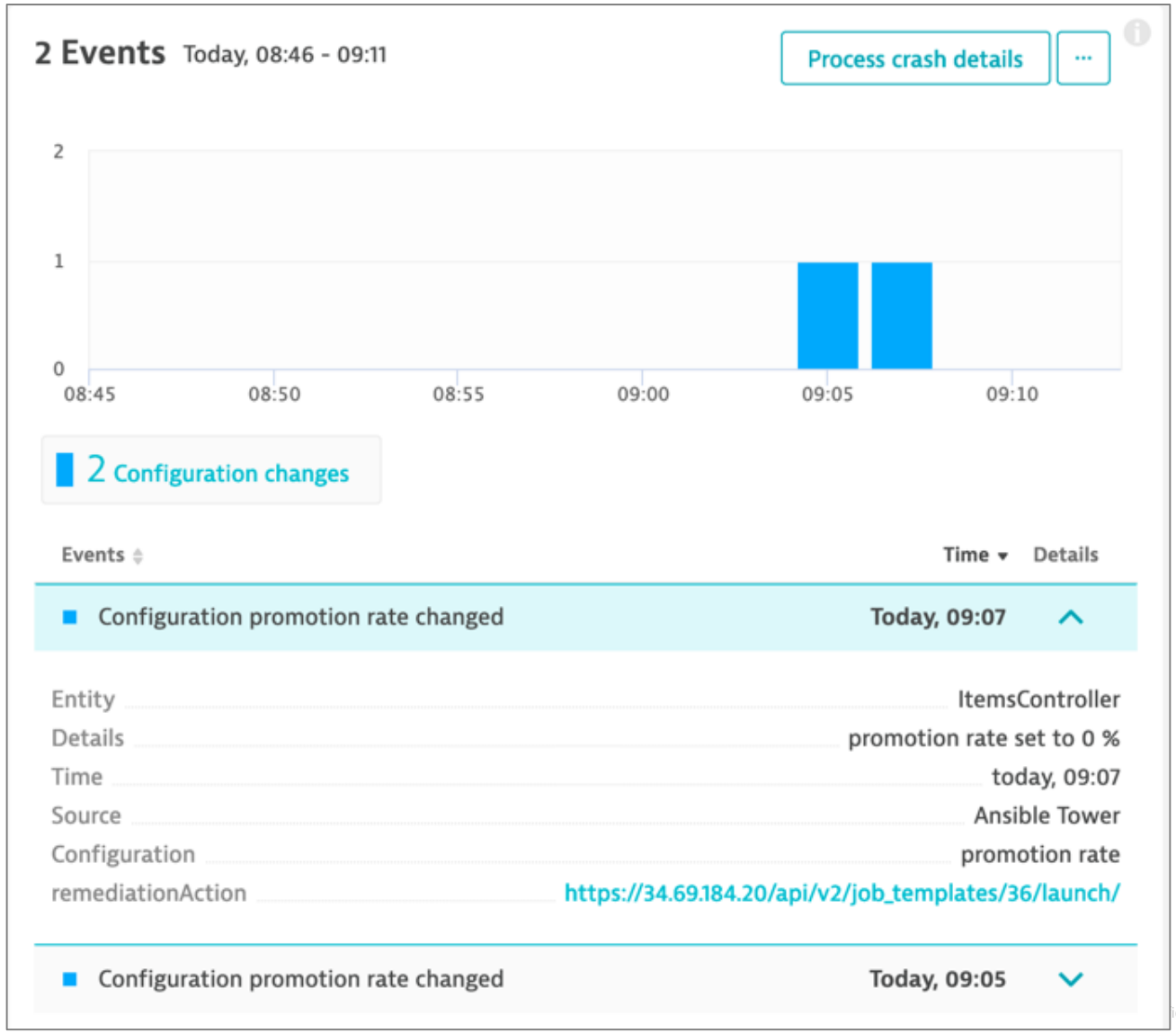
Observe the auto-remediation playbook actions

- Go back to the Service view
- The comments section will show the remediation actions taken by Ansible Tower



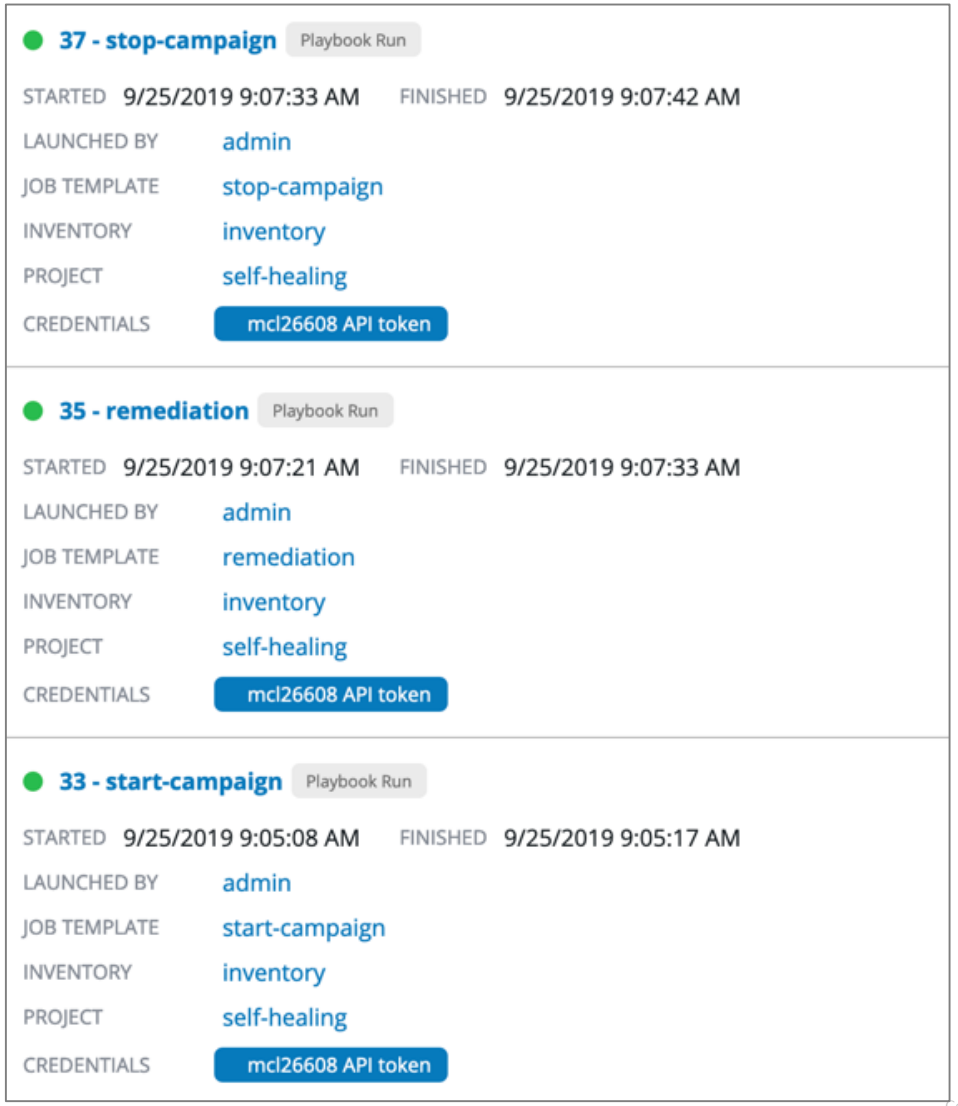
Observe the auto-remediation playbook actions

- Drill-down in the Problem.
- You will see a new configuration change event reported by Ansible Tower
- The promotional rate has been set back to 0% to remediate to the transaction failures



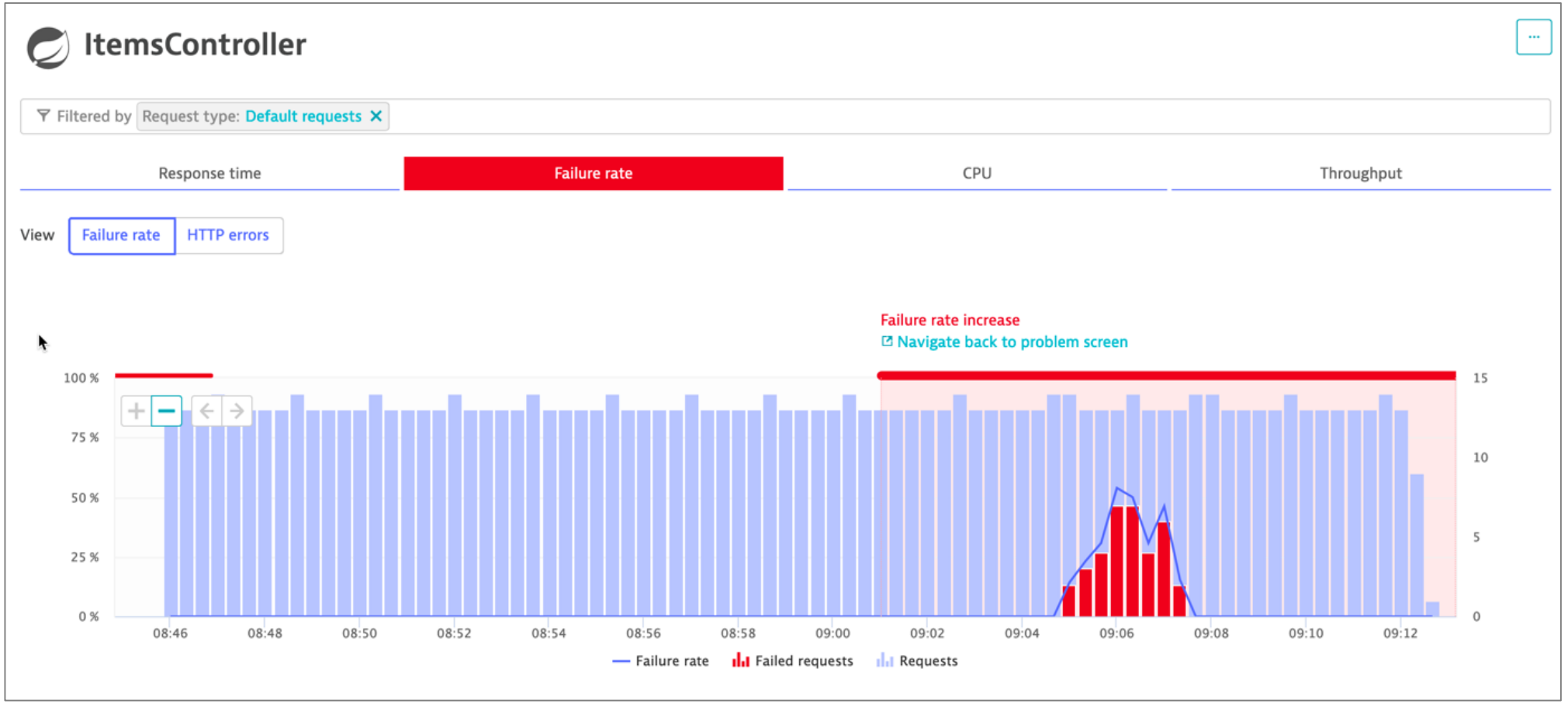
Observe the auto-remediation playbook actions

- Jobs executed in Ansible Tower
  - start-campaign (set rate to 50%)
  - remediation
    - push comment to Dynatrace Problem
    - retrieve problem details
    - launch remediation action related to problem context
    - update Dynatrace Problem
  - stop-campaign (set rate to 0%)

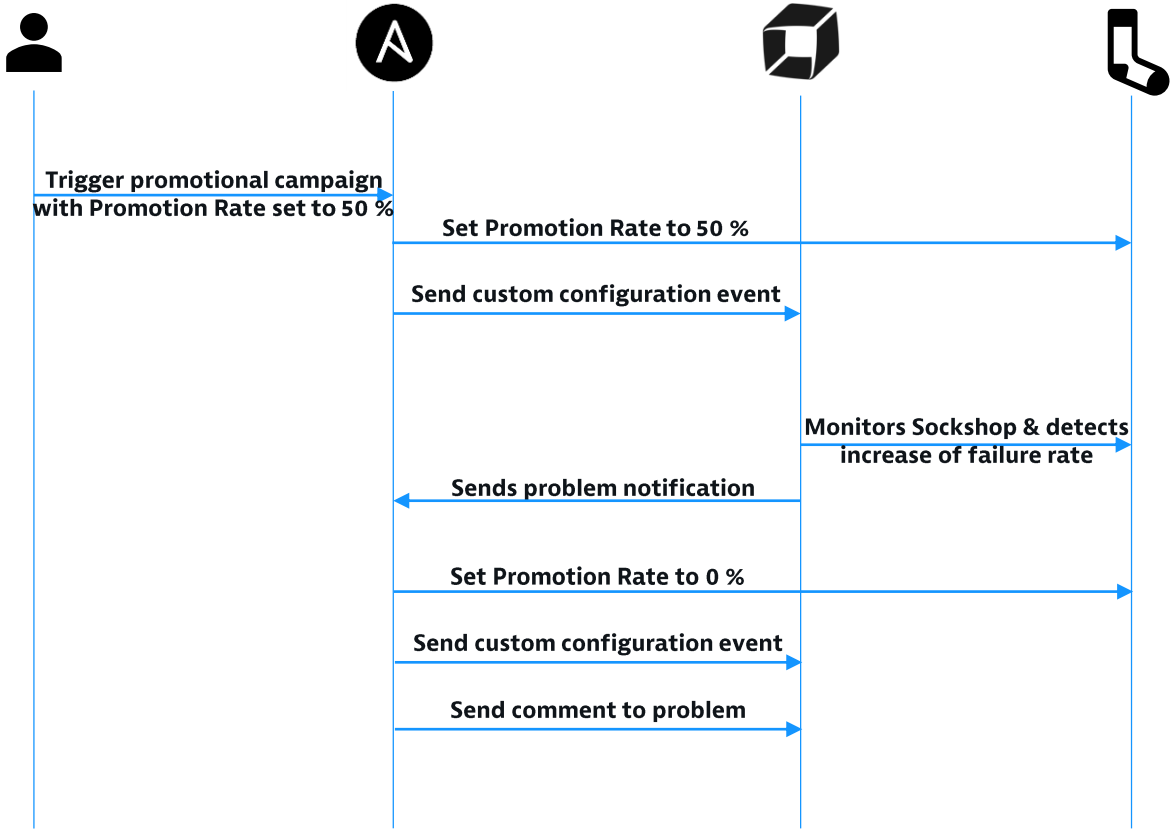


That's it! Problem solved automatically

- Look at the Service Failure Rate. It has returned to 0%



Runbook automation workflow recap



Clean up

Clean up

- If you want to keep the cluster running but:
  - remove Jenkins : execute the script **removeJenkins.sh** contained in the **utils** directory
  - remove Ansible Tower : execute the script **removeTower.sh** container in the **utils** directory

- If you want to delete everything, feel free to delete the GKE cluster via the GKE Console

dynatrace.com