Zero to Cloud-Native with IBM Cloud

Part 9C: Finish Deploying and Testing the Application

Kevin Collins
kevincollins@us.ibm.com
Technical Sales Leader
IBM Cloud Enterprise Containers – Americas

Kunal Malhotra
kunal.malhotra3@ibm.com
Cloud Platform Engineer
IBM Cloud MEA

1 Create remaining Pipelines

If you followed parts 9A and 9B, you created toolchains for the **frontend api** and **frontend web** microservices. Next, you will need to create additional pipelines for the following microservices:

- ocp-realtime-02cn
- enable-node-ssh-02cn
- load-ocp-versions-02cn
- utility-02cn

Choose your favorite method from with Classic or Tekton to create the remaining delivery pipelines.

After you have created all your delivery pipelines, make sure all of your pods are running. Log into your OpenShift cluster using the terminal and run:

```
oc get pods
```

```
→ oc get pods
NAME                                      READY   STATUS
api-frontend-02cn-5b65dcffcd-lrtzh        1/1     Running
enable-node-ssh-02cn-6bb66b46d8-78k5d     1/1     Running
enable-node-ssh-02cn-6bb66b46d8-z6cht     1/1     Running
load-ocp-versions-02cn-1599163740-rlz9h   0/1     Completed
load-ocp-versions-02cn-1599163800-pbzhg   0/1     Completed
load-ocp-versions-02cn-1599163860-6v7hr   0/1     Completed
ocp-realtime-02cn-5549744b96-m8smq        1/1     Running
utility-02cn-87f768c58-pjbts              1/1     Running
utility-02cn-87f768c58-rqdnh              1/1     Running
web-frontend-02cn-697d6875c-pgj9h         1/1     Running
```

Your output should look like the above. Note that the load-ocp-versions-02cn microservice is a cronjob that runs every 5 minutes. You may see a different number of completions.

2 – Configure Cloud Internet Service

Part of the deployment.yaml files that were run from the frontend api and web delivery pipelines created a load balancer service. RedHat OpenShift on IBM Cloud will automatically create a VPC load balancer when you indicate in your deployment file that you want to create a load balancer service.

What we will need to do is map the load balancer url to a DNS entry for the domain we created with IBM Cloud Internet Services. The first thing we need to do is find the load balancer external IP.
To do so, log into your OpenShift cluster using the terminal. Once you are logged in, switch to the zero-to-cloud-native projects and list all the services using the following commands:

```
oc project zero-to-cloud-native
oc get svc
```

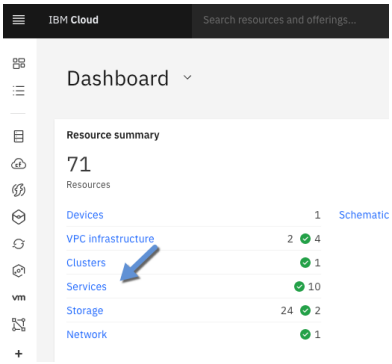
```
→ oc project zero-to-cloud-native
Already on project "zero-to-cloud-native" on server "https://c107-e.us-south.containers.cloud.ibm.com:31899".

enable-node-SSH-02cn on  master [!?] on  v19.03.8 at  zero-to-cloud-native/c107-e-us-south-containers-cloud-ibm-com:318
-to-cloud-native) took 2s
→ oc get svc
NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP                                PORT(S)          AGE
api-frontend-02cn-service   LoadBalancer   172.21.99.124    2f9437d3-us-south.lb.appdomain.cloud       8000:31306/TCP   6d23h
ocp-realtime-02cn-service   ClusterIP      172.21.84.97     <none>                                     8220/TCP         7d
utility-02cn-service        ClusterIP      172.21.69.193    <none>                                     8110/TCP         6d7h
web-frontend-02cn           LoadBalancer   172.21.161.160   d7629d42-us-south.lb.appdomain.cloud       8080:30035/TCP   21h
```

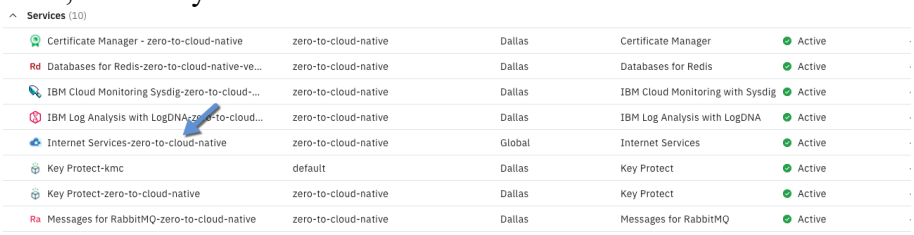
This will output 4 services.
api-frontend-02cn-service –external load balancer service for the api frontend.
ocp-realtime-02cn-service – this is an internal service for the ocp realtime microservice. This means the microservice is only available within the cluster. In our case, only the api-frontend will call this service.
utility-02cn-service – another internal service that provides common APIs that the other microservices will call.
web-frontend-02cn-service –external load balancer service for the web frontend.

Starting with the api frontend microservice, copy the external ip that you retrieved when you listed all of your services. In this case, **2f9437d3-us-south.lb.appdomain.cloud.** Note, you will have a different value.

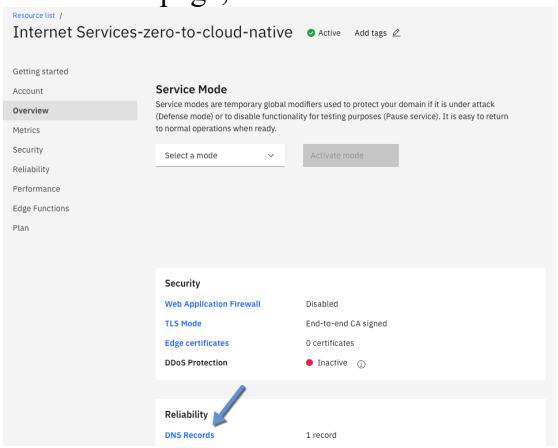
Next, we need to create a DNS CNAME in Cloud Internet Services, mapping this external ip to the domain we created. From you IBM Cloud dashboard click on Services.



Next, click on your Cloud Internet Services for the zero to cloud native tutorial.

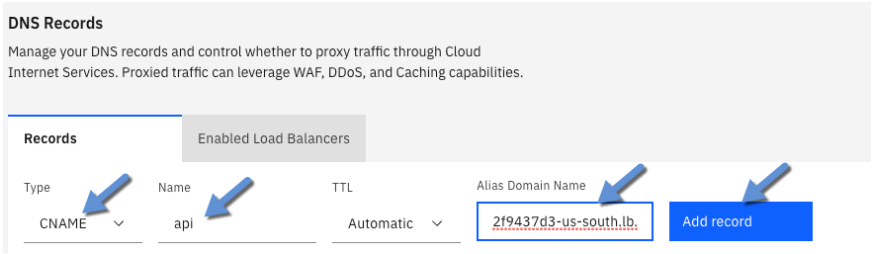


On the next page, click on DNS Records.

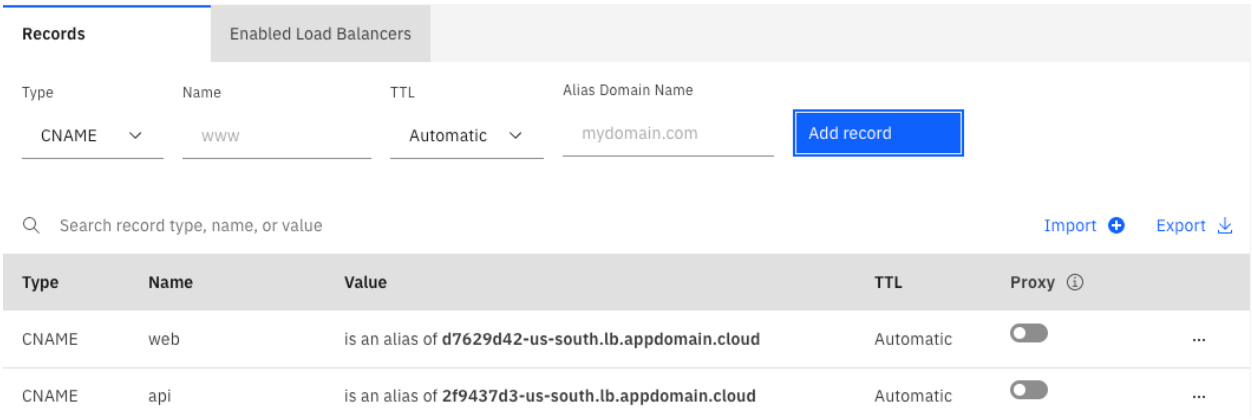


To create a DNS entry, we are going to create a CNAME DNS record.
Type: CNAME
Name: Past in the value you copied from the zero-to-cloud-native-api-service
Alias Doman: api.<the domain CIS is managing>

See an example below:



Repeat the same steps for web frontend service. After you are done, you should see two new DNS entries.



Congratulations!! You have now successfully deployed and configured the zero to cloud native application. Now let's test it!

3 – Test the application

As we have discussed, there are two frontend interfaces to the zero to cloud native tutorial application. A traditional web frontend and an API frontend.

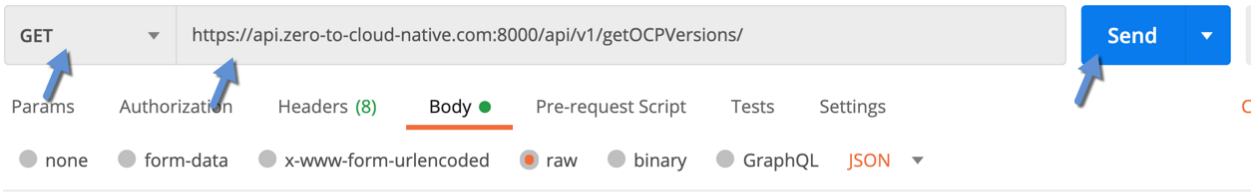
3 – 1 Testing the API Application

To test the API interface, I will be using Postman which you installed in Part 5. Feel free to use any other API tool or even the curl command if you like.

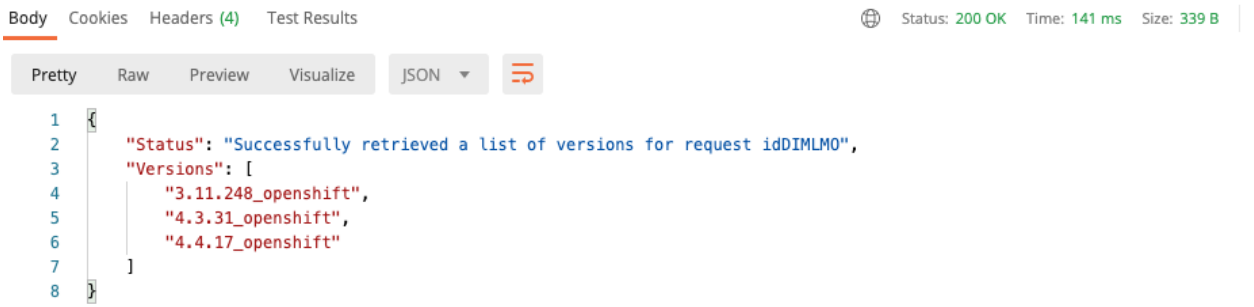
3 – 1 – 1 Get OCP Versions

Start by testing the get OCP Versions API. Open the PostMan application, change to a GET call, and enter your host name. If you followed the same naming conventions, your host name of this API should be api.<the domain you created in part 3>:8000/api/v1/getOCPVersions/

In my case, the host for this API is:
https://api.zero-to-cloud-native.com:8000/api/v1/getOCPVersions/



After entering this hostname, you will get a JSON response like this:



The output will contain the current version of RedHat OpenShift that are supported by IBM.

3 – 1 – 2 Get OCP Token

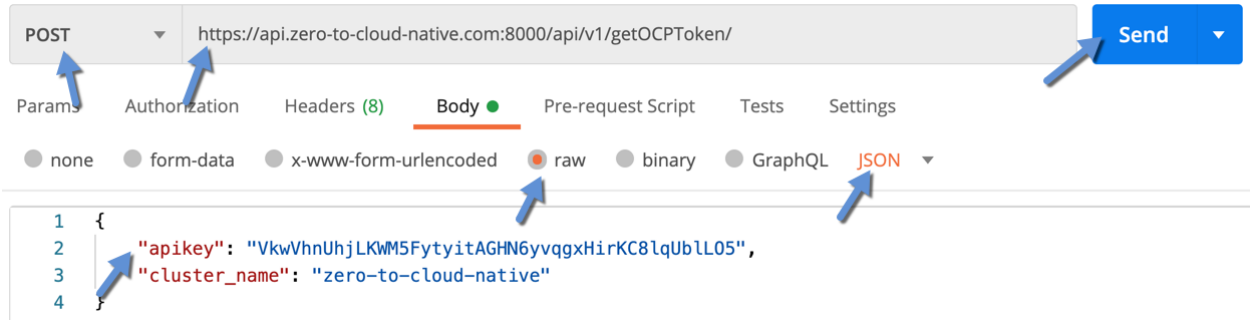
Next, we will test retrieving a token to log into OpenShift. Just like you did to test the get OCP versions API, select **POST**, keep the domain you had above, but change the endpoint to getOCPToken.

In my example, the url is:
`https://api.zero-to-cloud-native.com:8000/api/v1/getOCPToken/`

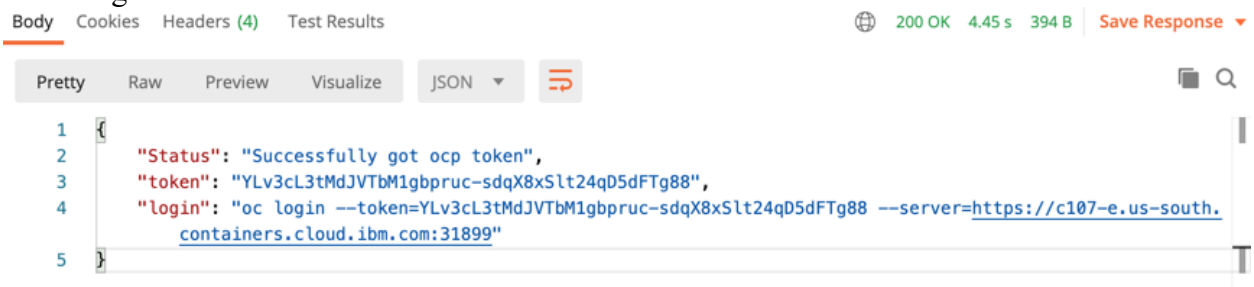
For this API, we will need to pass a couple of variables, an IBM Cloud API Key for the account where the cluster that you want to retrieve the OCP login token is in.

Under the Body tab, click on raw and on the right, select JSON. Then in the input window enter JSON in this format:

```
{
    "apikey": "<IBM CLOUD API KEY",
    "cluster_name": "e.g. zero-to-cloud-native"
}
```



After entering all of these settings, click Send. This will return a JSON response like the following:



Now, let's test the login command to see if we can log into our OpenShift cluster. Copy the login field:

```
oc login --token=YLv3cL3tMdJVTbM1gbpruc-sdqX8xSlt24qD5dFTg88 --
server=https://c107-e.us-south.containers.cloud.ibm.com:31899
```

Start your iTerm terminal and paste the command and hit enter.

```
→ oc login --token=YLv3cL3tMdJVTbM1gbpruc-sdqX8xSlt24qD5dFTg88 --server=https://c107-e.us-south.containers.cloud.ibm.com:31899
Logged into "https://c107-e.us-south.containers.cloud.ibm.com:31899" as "IAM#kevincollins@us.ibm.com" using the token provided.

You have access to 60 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "zero-to-cloud-native".
```

You should see output like the above. Now when you want to log into your OpenShift cluster, you can just run this API and don't have to log into the IBM Cloud and then the OpenShift cluster.

Note, you can also call the API using curl from the command line.
Example:

```
curl -X POST -H "Content-Type: application/json" "https://api.zero-to-cloud-native.com:8000/api/v1/getOCPToken/" -d '{"apikey": "IFvxsWmtv0r3IDDHJAJAHJHAJHJHJHHGres24uByKbc12H", "cluster_name": "zero-to-cloud-native"}'
```

```
ocp-realtime-02cn on  master [!?] on  v19.03.8 at  zero-to-cloud-native/c107-e-us-south-containers-cloud-ibm-com:31899/IAM#kevincollins@us.ibm.com (zero-to-cloud-native)
→ curl -X POST -H "Content-Type: application/json" "api.zero-to-cloud-native.com:8000/api/v1/getOCPToken/" -d '{"apikey": "IFvxsWmtv0r3IDRgLeoRWuEHJeAzwGres24uByKbc12H", "cluster_name": "zero-to-cloud-native"}'
{
    "Status": "Successfully got ocp token",
    "token": "0fdVYAtCXplcnMUSAXRSkFkKSu8SR1Wrq9YvYpUk2Q4",
    "login": "oc login --token=0fdVYAtCXplcnMUSAXRSkFkKSu8SR1Wrq9YvYpUk2Q4 --server=https://c107-e.us-south.containers.cloud.ibm.com:31899"
}
```

Note, the example above uses a temporary API that has been removed.

3 – 1 – 3 Enable SSH

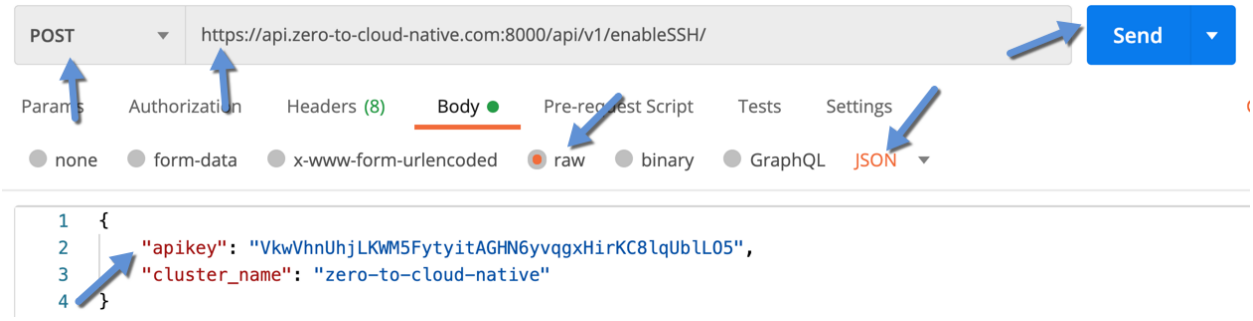
Generally speaking, we don't recommend SSHing directly into OpenShift worker nodes, however when you are doing advanced configuration and debugging, you might find that you need to. With IBM's Managed OpenShift Service, you can but there are a number of steps that you have to take that are rather cumbersome. You can find the manual instructions here: https://developer.ibm.com/technologies/containers/tutorials/ssh-connect-nodes-kubernetes-cluster-openshift/

As I find myself doing this frequently, I have automated it through the enableSSH API. Let's test it.

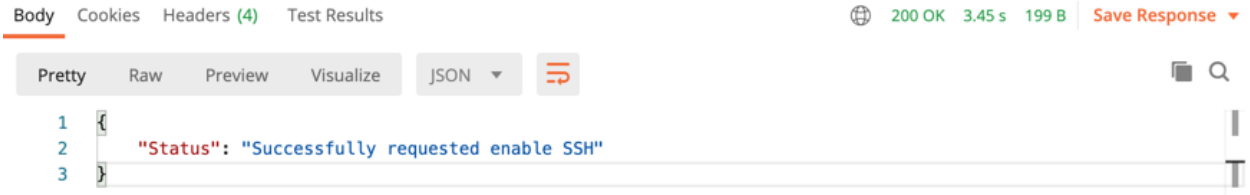
Following the same steps as we did with get OCP Token, you will select POST, change the endpoint to enableSSH. The JSON input take the same parameters, your IBM Cloud apikey and the cluster_name.

https://api.zero-to-cloud-native.com:8000/api/v1/enableSSH/

```
{
    "apikey": "<IBM CLOUD API KEY",
    "cluster_name": "e.g. zero-to-cloud-native"
}
```



After entering these settings, click Send. You will get output like the following:



Note, this API is long running (~10 seconds). Following the CQRS Command microservices pattern, the output will indicate that the request to enable SSH has been accepted. The reason for this is we don't want the frontend API to block other requests while this request completes. In the logging section, we will go through how you and your users can find out the status of the request.

For now, let's test if we can in fact SSH into a worker node. What the API will do, is it will created 'inspect' pods that will give us access to the underlying worker node the pod is running on.

Using **iTerm2**, log into your OpenShift cluster. Use the getToken API if you like! After logging in, run:

```
oc get pods -n kube-system
```

```
→ oc get pods -n kube-system
NAME                                                READY   STATUS    RESTARTS   AGE
ibm-keepalived-watcher-52zfc                        1/1     Running   0          2d
ibm-keepalived-watcher-7nq95                        1/1     Running   0          2d
ibm-keepalived-watcher-dqs54                        1/1     Running   0          2d
ibm-keepalived-watcher-kv975                        1/1     Running   0          2d1h
ibm-keepalived-watcher-l2vzm                        1/1     Running   0          2d1h
ibm-keepalived-watcher-ntsvf                        1/1     Running   0          2d
ibm-keepalived-watcher-sspgt                        1/1     Running   0          2d1h
ibm-keepalived-watcher-vpqq5                        1/1     Running   0          2d
ibm-keepalived-watcher-w676s                        1/1     Running   0          2d
ibm-master-proxy-static-10.240.0.4                  2/2     Running   0          2d1h
ibm-master-proxy-static-10.240.0.5                  2/2     Running   0          2d
ibm-master-proxy-static-10.240.0.6                  2/2     Running   0          2d
ibm-master-proxy-static-10.240.128.4                2/2     Running   0          2d
ibm-master-proxy-static-10.240.128.5                2/2     Running   0          2d
ibm-master-proxy-static-10.240.128.6                2/2     Running   0          2d
ibm-master-proxy-static-10.240.64.4                 2/2     Running   0          2d
ibm-master-proxy-static-10.240.64.5                 2/2     Running   0          2d
ibm-master-proxy-static-10.240.64.6                 2/2     Running   0          2d
ibm-vpc-block-csi-controller-0                      4/4     Running   0          2d1h
ibm-vpc-block-csi-node-29bt7                        3/3     Running   0          2d
ibm-vpc-block-csi-node-2njgf                        3/3     Running   0          2d
ibm-vpc-block-csi-node-7zpvf                        3/3     Running   0          2d1h
ibm-vpc-block-csi-node-g5fgh                        3/3     Running   0          2d1h
ibm-vpc-block-csi-node-j7grb                        3/3     Running   0          2d1h
ibm-vpc-block-csi-node-pxz8w                        3/3     Running   0          2d
ibm-vpc-block-csi-node-tdjds                        3/3     Running   0          2d
ibm-vpc-block-csi-node-xs5tb                        3/3     Running   0          2d
ibm-vpc-block-csi-node-zdmvs                        3/3     Running   0          2d
inspectnode-6sqohaugmvg-zerotocloud-default-00000a63   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00000bde   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00000c56   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00000d7d   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00000e16   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00000f8f   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-000010f0   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00001122   1/1     Running   0          42h
inspectnode-6sqohaugmvg-zerotocloud-default-00001278   1/1     Running   0          42h
vpn-898f644bd-c5w5v                                 1/1     Running   0          2d
```

Note the nodes that start with inspect. You will see the pods follow the naming convention of inspectnode-<cluster id>-<cluster name>-<worker pool>-<worker id>.
Following this naming convention will allow us to easily identify which pod is associated with which worker node.

Now let test it. Pick a pod and copy the pod name, I will choose **`inspectnode-6sqohaugmvg-zerotocloud-default-00000a63`**

Next, we are going to 'exec' in the pod by running this command. Make sure to use the name of the pod you just copied.

```
kubectl exec -it inspectnode-6sqohaugmvg-zerotocloud-default-00000a63 /bin/sh
-n kube-system
```

Your prompt should now change to # … indicating you are now connected inside the inspect pod.

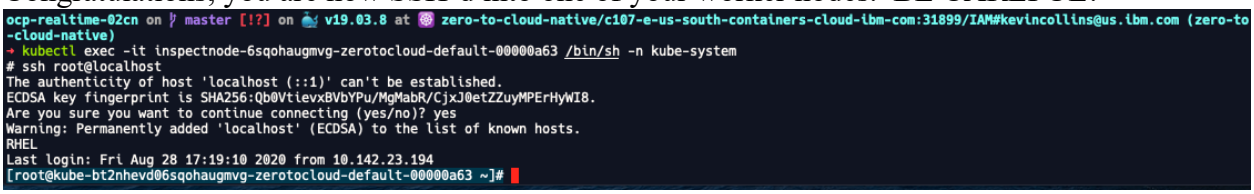
Next, to SSH in the worker node type:

```
ssh root@localhost
```

You should see your prompt now change to something like this:

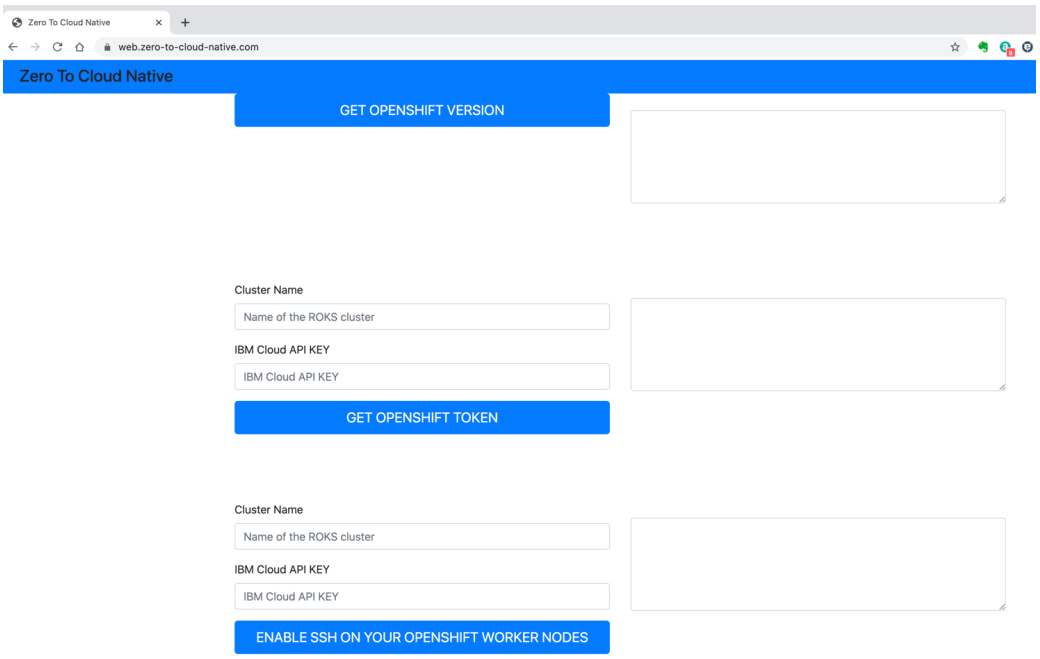
```
[root@kube-bt2nhevd06sqohaugmvg-zerotocloud-default-00000a63 ~]#
```

Congratulations, you are now SSH'd into one of your worker nodes. BE CAREFUL!

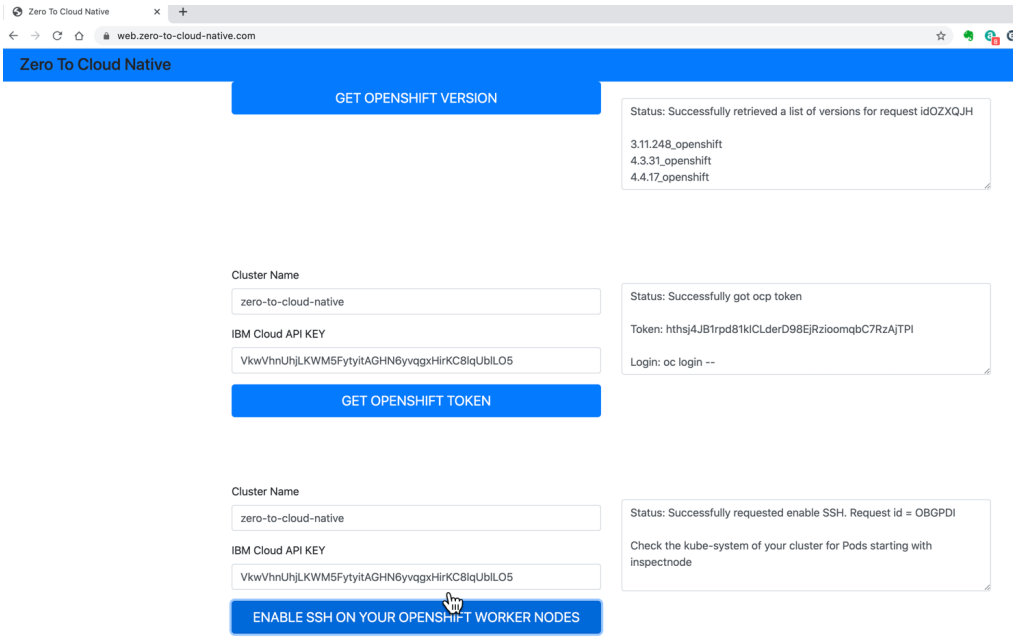


4 – Testing the Web Application

The last step we want to do is test the web frontend of the application. To do so navigate to: web.<your domain>. In my case web.zero-to-cloud-native.com



Enter your apikey and cluster name for the Get OpenShift Token and Enable SSH APIs and click each api to execute the api and get a result.



5 – Next Up

If you have made it this far, congratulations you have successfully deployed a cloud-native application end-to-end. I hope you have learned a lot and have found this tutorial to be useful. This completes phase 1 or in other words day 1.

Stay tuned for future sessions that will go through day two operations like logging, monitoring, deploying to a remote location, automating and optimizing the environment. This will be a living series where we will be adding new sessions covering a variety of topics. As always, we would love your feedback and suggestions on future sessions.