Zero to Cloud-Native with IBM Cloud

Part 10A: Export Logs

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

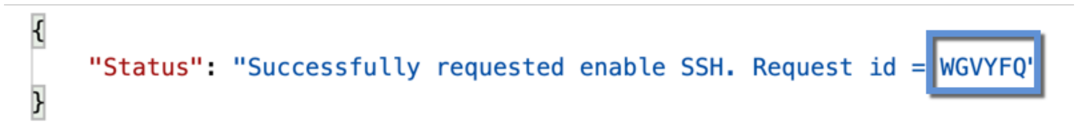
1 - Introduction

In part 10 of this tutorial series, you created a logging dashboard with several logging views and graphical representations of both APIs and Microservices that make up our application. In this part, we will be exporting specific log messages so that our users can understand what is happening with their long running API requests.

1 – 1 Enable Node SSH

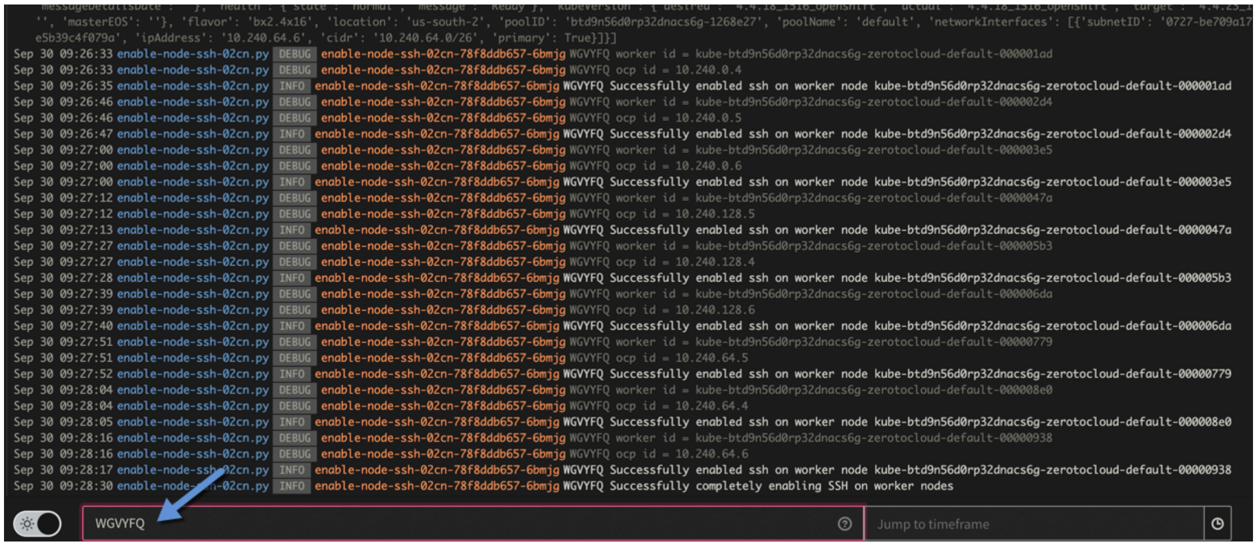
Enable Node SSH is our long running API task that will immediately return that the request to enable ssh on worker nodes was received. Since it is a long running task, we don't want to block other requests for this one to complete.

How can users determine if their request was actually successful? One easy way is with LogDNA! We can export relevant logging lines to a user so they can see what is happening in the backend. As you will remember, when you submit the request to enable SSH a request id is returned.



```
{
    "Status": "Successfully requested enable SSH. Request id = WGVYFQ'
}
```

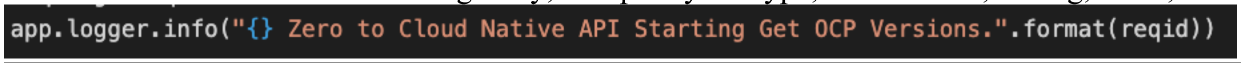
Using that Request id, we can go to our logDNA dashboard and search for that request id to see what happened.



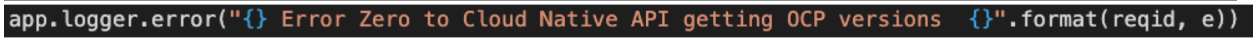
```
   '', 'masterEOS': ''}, 'flavor': 'bx2.4x16', 'location': 'us-south-2', 'poolID': 'btd9n56d0rp32dnacs6g-1268e27', 'poolName': 'default', 'networkInterfaces': [{'subnetID': '0727-be709a17
   e5b39c4f079a', 'ipAddress': '10.240.64.6', 'cidr': '10.240.64.0/26', 'primary': True}]}]
Sep 30 09:26:33 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000001ad
Sep 30 09:26:33 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.0.4
Sep 30 09:26:35 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000001ad
Sep 30 09:26:46 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000002d4
Sep 30 09:26:46 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.0.5
Sep 30 09:26:47 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000002d4
Sep 30 09:27:00 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000003e5
Sep 30 09:27:00 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.0.6
Sep 30 09:27:00 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000003e5
Sep 30 09:27:12 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-0000047a
Sep 30 09:27:12 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.128.5
Sep 30 09:27:13 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-0000047a
Sep 30 09:27:27 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000005b3
Sep 30 09:27:27 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.128.4
Sep 30 09:27:28 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000005b3
Sep 30 09:27:39 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000006da
Sep 30 09:27:39 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.128.6
Sep 30 09:27:40 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000006da
Sep 30 09:27:51 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-00000779
Sep 30 09:27:51 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.64.5
Sep 30 09:27:52 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-00000779
Sep 30 09:28:04 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000008e0
Sep 30 09:28:04 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.64.4
Sep 30 09:28:05 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000008e0
Sep 30 09:28:16 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ worker id = kube-btd9n56d0rp32dnacs6g-zerotocloud-default-00000938
Sep 30 09:28:16 enable-node-ssh-02cn.py DEBUG enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ ocp id = 10.240.64.6
Sep 30 09:28:17 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-00000938
Sep 30 09:28:30 enable-node-ssh-02cn.py INFO enable-node-ssh-02cn-78f8ddb657-6bmjg WGVYFQ Successfully completely enabling SSH on worker nodes
```

You can see here a number of messages both at the Debug and Info level.

When our microservices create a log entry, we specify the type; such as Info, Debug, Error, etc



```
app.logger.info("{} Zero to Cloud Native API Starting Get OCP Versions.".format(reqid))
```



```
app.logger.error("{} Error Zero to Cloud Native API getting OCP versions  {}".format(reqid, e))
```

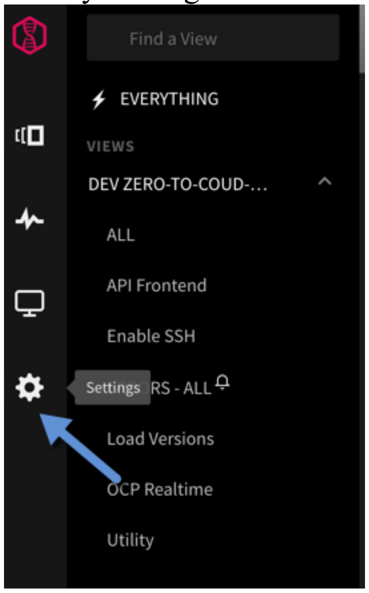
By having different levels of logging messages we can then filter which level you want to see. In our case, if we are sharing log messages with end users, we will only want to show them the high-level logging (INFO) messages.

2 – Customizing LogDNA to Export Logs

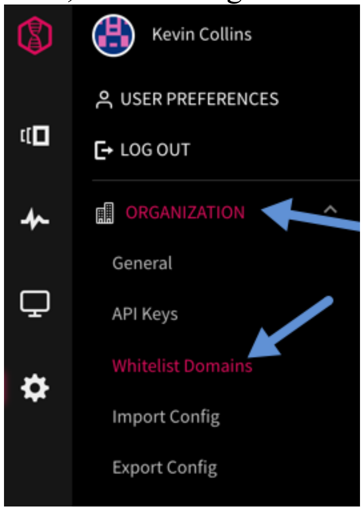
Before we can actually export logs from LogDNA, there are a couple of things we need to do with LogDNA. The first thing is to add the domain you created to a list of whitelisted domains that have access to LogDNA.

2 -1 Whitelist your Domain

From your LogDNA dashboard, click on the settings Icon.

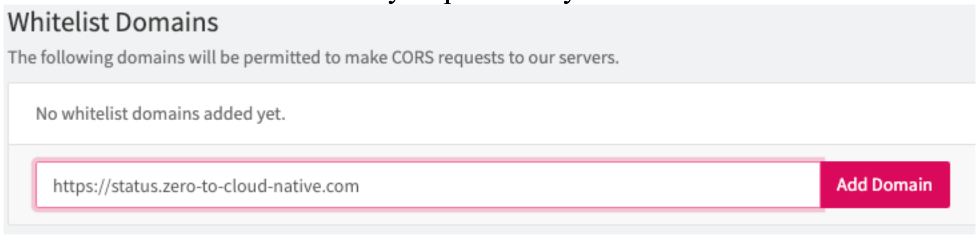


Next, click on Organization and then Whitelist Domains.



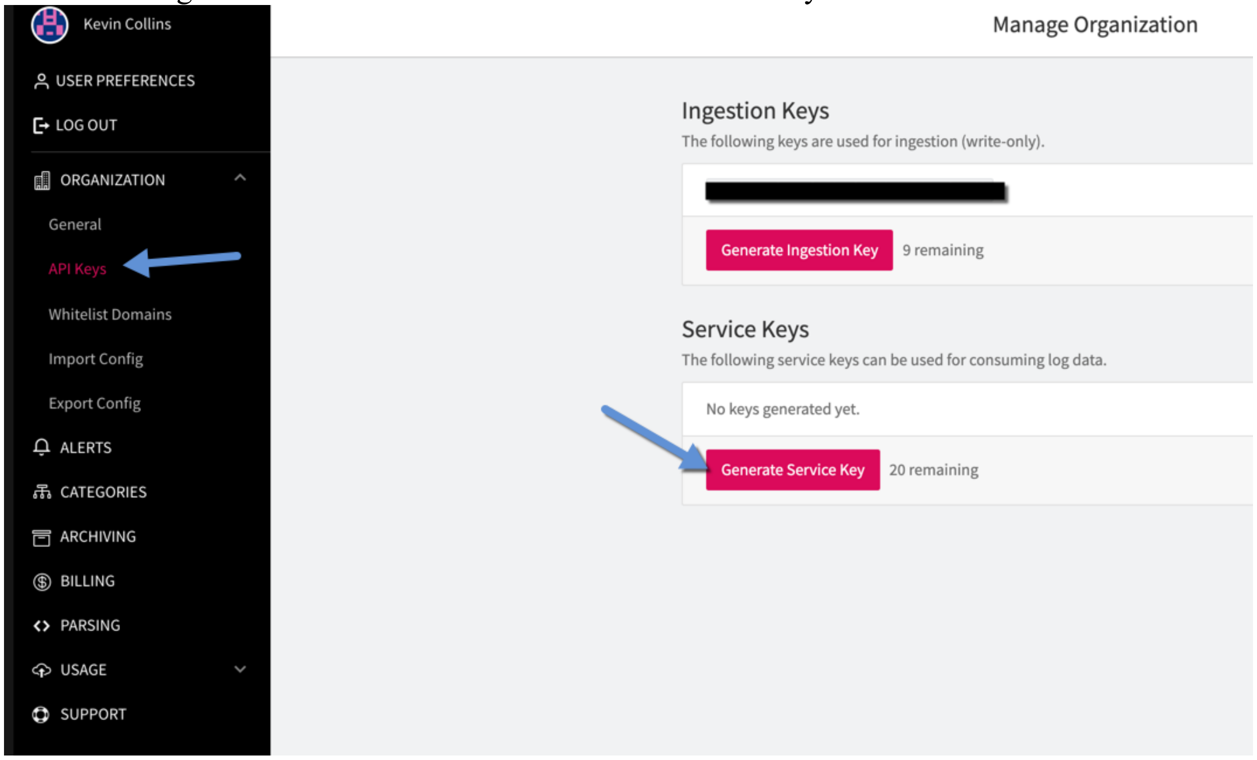
Next, you will need to add the domain you created to a list of domains that will permitted to make requests to LogDNA servers.

I will add the domain and the status application we will deploy next, **https://status.zero-to-cloud-native.com** and then click on **Add Domain**. Note: you will need to substitute zero-to-cloud-native with the domain you previously created.


Whitelist Domains
The following domains will be permitted to make CORS requests to our servers.
No whitelist domains added yet.
https://status.zero-to-cloud-native.com
Add Domain

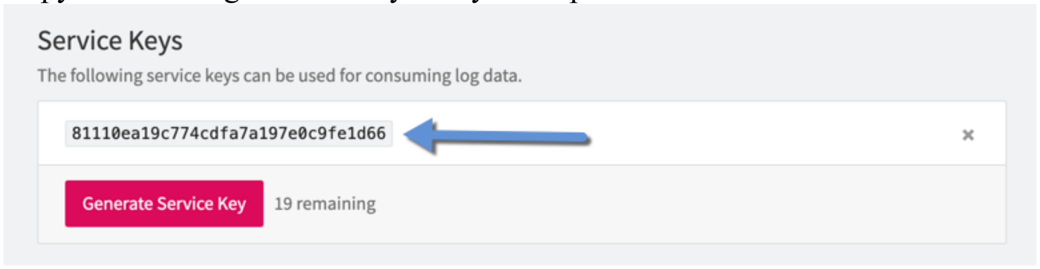
2 -2 Create a Service Key

Next, we will need to get a service key for our LogDNA instance. To do, click on API Keys from the navigation tab and then click on Generate Service Key.



[redacted]

Copy the resulting Service Key into your clipboard.

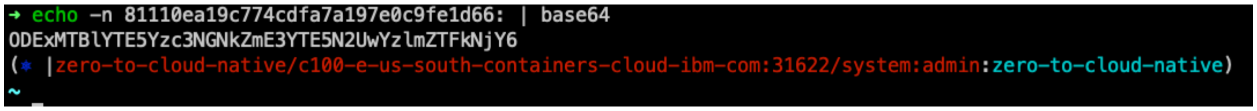

Service Keys
The following service keys can be used for consuming log data.
Generate Service Key
19 remaining

```
81110ea19c774cdfa7a197e0c9fe1d66
```

2-3 Base64 Encode Service Key

Next, we need to convert this key to base64. Start iTerm2 and run the following command. Note: make sure to add a : at the end of service key you copied.

```
echo -n <logdna service key>: | base64
```



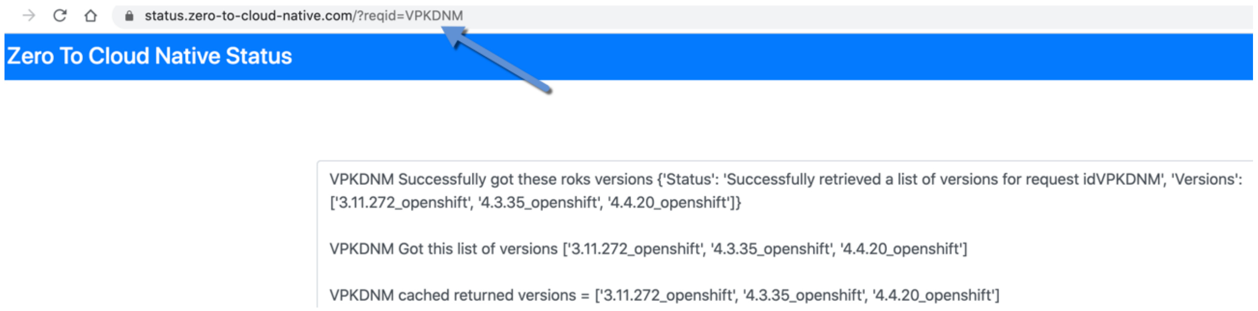
```
→ echo -n 81110ea19c774cdfa7a197e0c9fe1d66: | base64
ODExMTBlYTE5Yzc3NGNkZmE3YTE5N2UwYzlmZTFkNjY6
(⎈ |zero-to-cloud-native/c100-e-us-south-containers-cloud-ibm-com:31622/system:admin:zero-to-cloud-native)
~
```

Copy the result in a safe place, you will need this later.

3 – Status Application

In order to provide a way for users to view log messages, we will need to first create an additional microservice / web application. This application will simply show log messages after being given a request id. We will then update the API application to include a link to the log messages after the API has been called.

This is a view of the status app. You can see the request id was given as a parameter in the URL.


status.zero-to-cloud-native.com/?reqid=VPKDNM
Zero To Cloud Native Status

```
VPKDNM Successfully got these roks versions {'Status': 'Successfully retrieved a list of versions for request idVPKDNM', 'Versions':
['3.11.272_openshift', '4.3.35_openshift', '4.4.20_openshift']}

VPKDNM Got this list of versions ['3.11.272_openshift', '4.3.35_openshift', '4.4.20_openshift']

VPKDNM cached returned versions = ['3.11.272_openshift', '4.3.35_openshift', '4.4.20_openshift']
```

This is just a very simply application to give you a feel for what we can do with exporting logs with LogDNA.

3 – 1 Cloning the Status Repository

Status-02cn – is the status web application that will return 'Info' log messages when given a request id. You can find the GitHub repository here:
https://github.com/kmcolli/status-02cn

Following the same steps in Part 8, section 2 of this tutorial series, we will first clone the repository and create our own git repository.

Start your iTerm2 terminal and make sure you are in the directory you created that contains all of your git repositories.

Next, clone the status repository.

```
git clone  https://github.com/kmcolli/status-02cn
```

Go to github.com and create a new repository called **status-02cn**. You can view the directions if needed in Part 8, Section 2-2.

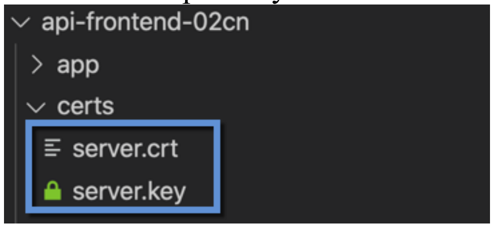
Import the code you just cloned into this new repository and then create a new **'dev' branch.** If you need a refresher on doing this, refer to the instructions in Part 8 – Section 5

Now that we have the source code in a dev branch, we can create a CI/CD pipeline for this new micro-service web application, but before we push the code with our CI/CD pipeline, we need to customize this microservice a little.

3 – 2 Customizing the Status Microservice

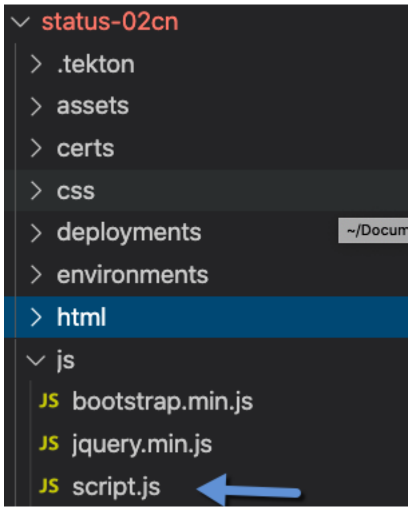
Because the status application is web facing, the next thing you will need to do is add the certificates for the domain you created to the nginx container that the status microservice uses.

To do so, just **copy the certificates** from api-frontend-02cn to the certs directory under the status-02cn repository.



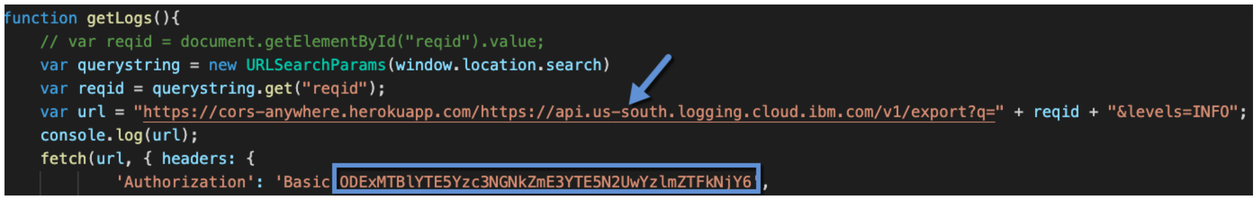
Next, we need to update the **js/script.js** file in the status-02cn repository. To do so, open Visual Studio Code and open the script.js file in the status-02cn repository.

Note: If you haven’t already, you will need to add the status-02cn folder to your workspace like you did for the other repositories.



Depending on where you deployed your LogDNA instance, you may need to update the url that will be called. If you deployed in Dallas, you can leave the URL as is that refers to the us-south region.

You will need to update the value in the blue box below. This value should be the base64 encoded value you retrieved in step 2-3.



```
function getLogs(){
    // var reqid = document.getElementById("reqid").value;
    var querystring = new URLSearchParams(window.location.search)
    var reqid = querystring.get("reqid");
    var url = "https://cors-anywhere.herokuapp.com/https://api.us-south.logging.cloud.ibm.com/v1/export?q=" + reqid + "&levels=INFO";
    console.log(url);
    fetch(url, { headers: {
            'Authorization': 'Basic ODExMTBlYTE5Yzc3NGNkZmE3YTE5N2UwYzlmZTFkNjY6',
```

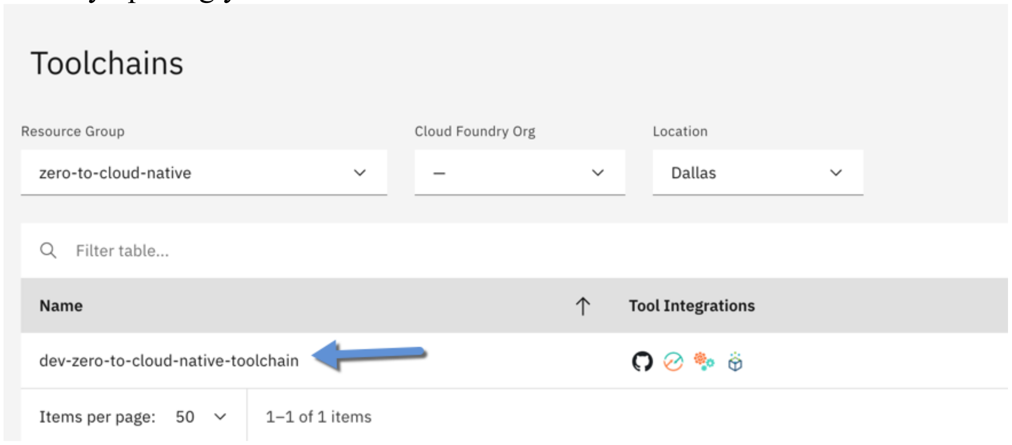
4 - Create Status CI/CD pipeline

Following best practices, we will create a CI/CD pipeline for this new micrservice. To create this new CI/CD pipeline, choose from either a classic pipeline (Part 9A) or a Tekton pipeline (Part 9B).

I will be creating a Tekton pipeline and will provide the high-level instructions here. If you need a refresher on the entire process, refer to Part 9A of this tutorial series.

4 - 1

Start by opening your **dev-zero-to-cloud-native toolchain**.



Toolchains

Resource Group: zero-to-cloud-native

Cloud Foundry Org: –

Location: Dallas

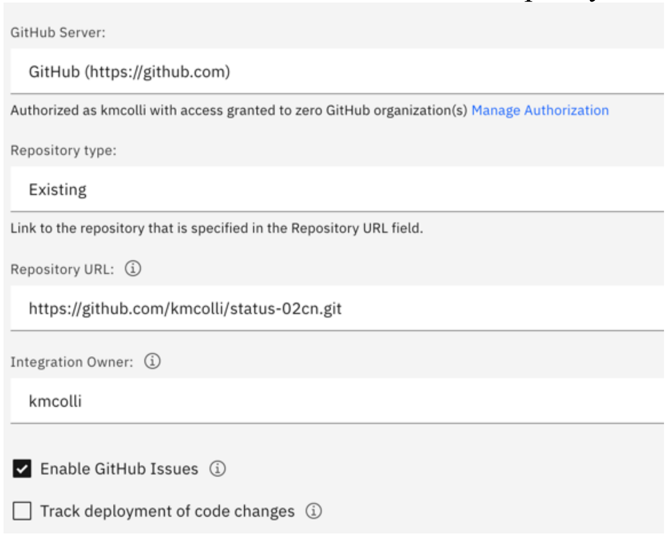
Filter table...

Name	Tool Integrations
dev-zero-to-cloud-native-toolchain	

Items per page: 50   1–1 of 1 items

4 - 2

Add a GitHub tool to the toolchain and specify the **status-02cn** repository you just created.



GitHub Server:

GitHub (https://github.com)

Authorized as kmcolli with access granted to zero GitHub organization(s) Manage Authorization

Repository type:

Existing

Link to the repository that is specified in the Repository URL field.

Repository URL:

https://github.com/kmcolli/status-02cn.git

Integration Owner:

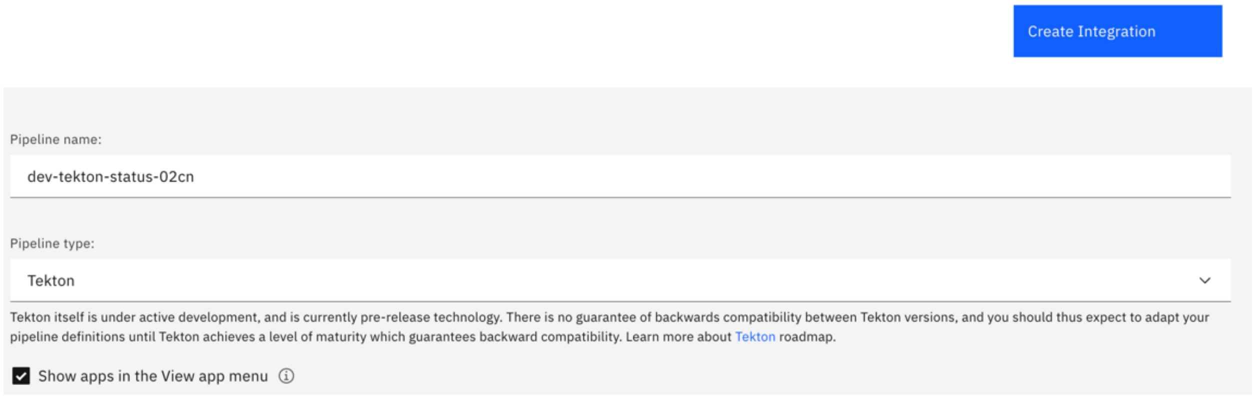
kmcolli

☑ Enable GitHub Issues

☐ Track deployment of code changes

4 - 3

Add a Delivery Pipeline tool to your toolchain, giving it the name of **dev-tekton-status-02cn** and type **Tekton**.



Pipeline name: dev-tekton-status-02cn

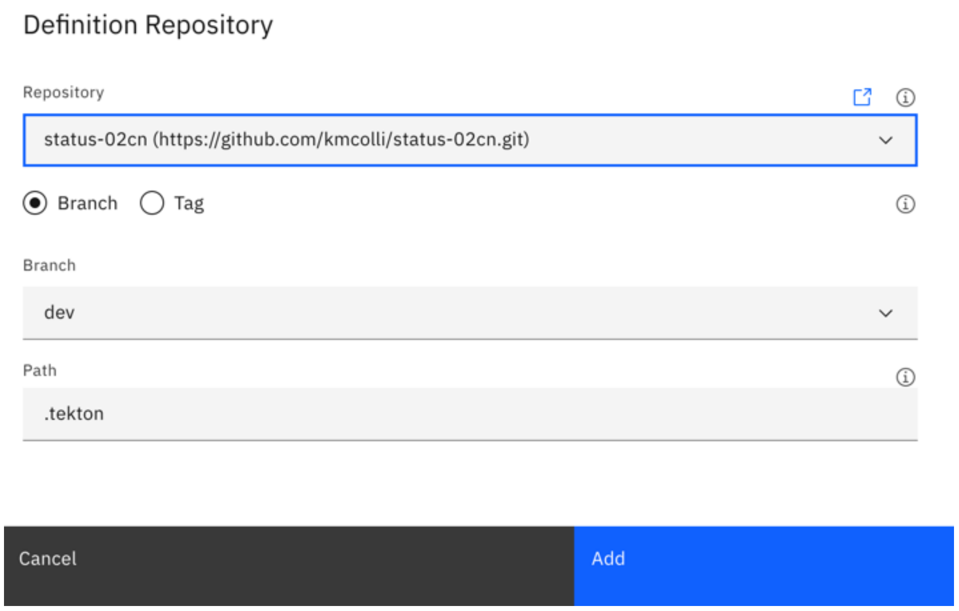
Pipeline type: Tekton

Tekton itself is under active development, and is currently pre-release technology. There is no guarantee of backwards compatibility between Tekton versions, and you should thus expect to adapt your pipeline definitions until Tekton achieves a level of maturity which guarantees backward compatibility. Learn more about Tekton roadmap.

[✓] Show apps in the View app menu

4 -4

Specify the definition of the repository by selecting the **dev branch** of the **status-02cn repository** you created.



Definition Repository

Repository: status-02cn (https://github.com/kmcolli/status-02cn.git)

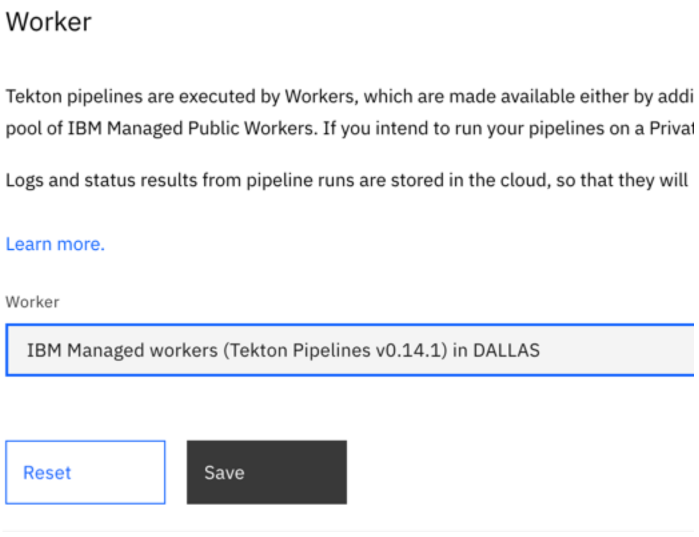
[✓ Branch] Tag

Branch: dev

Path: .tekton

4 - 5

Select the **IBM Cloud Managed Worker**.



Worker

Tekton pipelines are executed by Workers, which are made available either by addi
pool of IBM Managed Public Workers. If you intend to run your pipelines on a Privat

Logs and status results from pipeline runs are stored in the cloud, so that they will

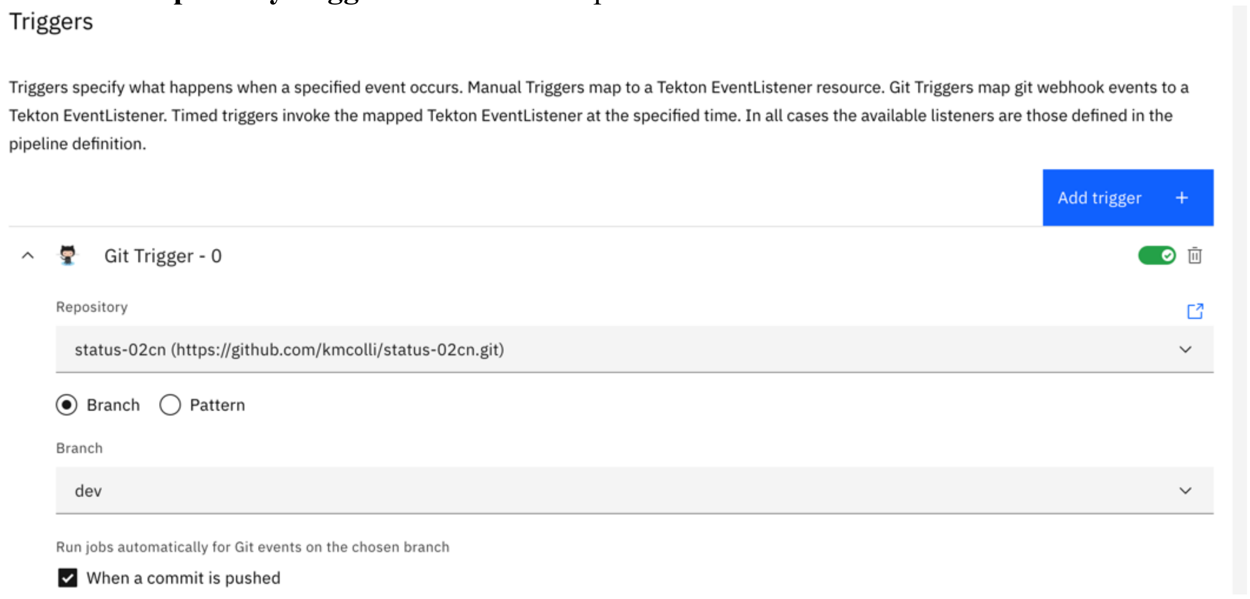
Learn more.

Worker

IBM Managed workers (Tekton Pipelines v0.14.1) in DALLAS

4 - 6

Add a **Git Repository trigger** for the status repo and **dev branch**.



Triggers

Triggers specify what happens when a specified event occurs. Manual Triggers map to a Tekton EventListener resource. Git Triggers map git webhook events to a Tekton EventListener. Timed triggers invoke the mapped Tekton EventListener at the specified time. In all cases the available listeners are those defined in the pipeline definition.

Git Trigger - 0

Repository

status-02cn (https://github.com/kmcolli/status-02cn.git)

[✓ Branch] Pattern

Branch

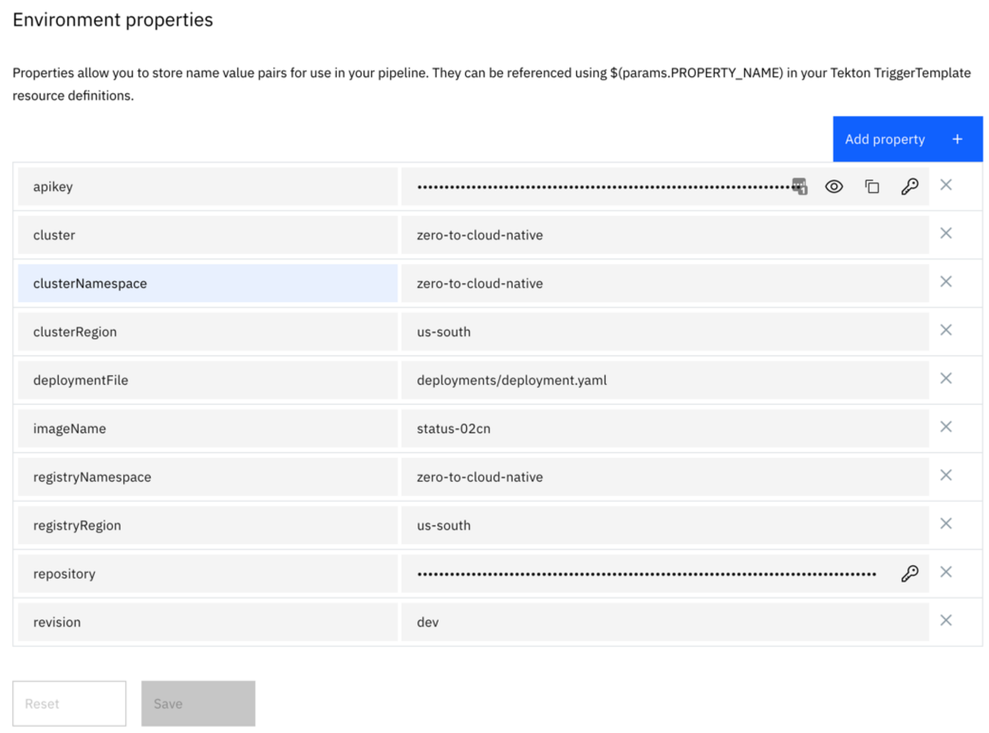
dev

Run jobs automatically for Git events on the chosen branch

[✓] When a commit is pushed

4 - 7

Enter your **environment variables**.



Environment properties

Properties allow you to store name value pairs for use in your pipeline. They can be referenced using $(params.PROPERTY_NAME) in your Tekton TriggerTemplate resource definitions.

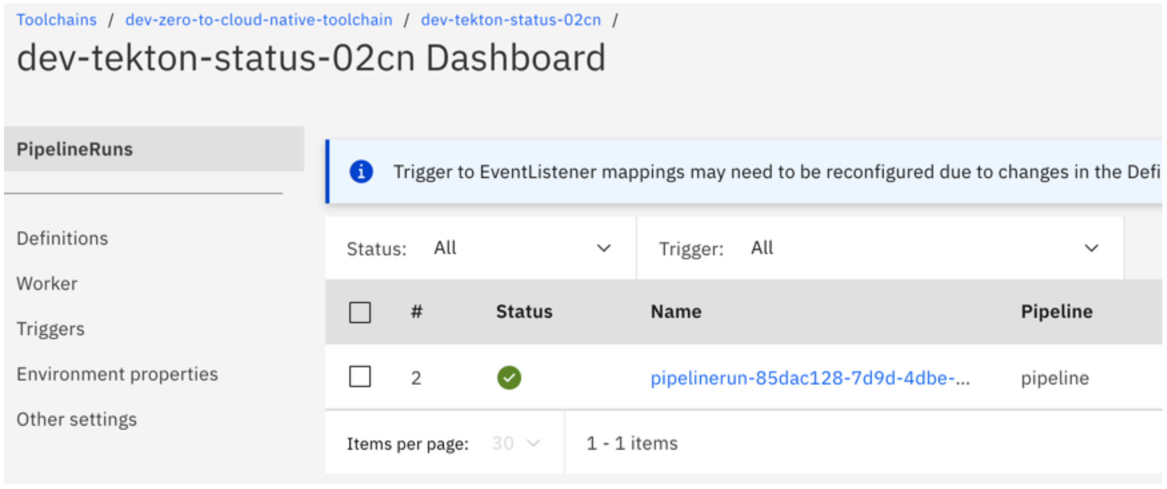
Add property +

apikey	••
cluster	zero-to-cloud-native
clusterNamespace	zero-to-cloud-native
clusterRegion	us-south
deploymentFile	deployments/deployment.yaml
imageName	status-02cn
registryNamespace	zero-to-cloud-native
registryRegion	us-south
repository	••
revision	dev

Reset Save

4 - 8

Commit a change to your GitHub repo and watch the pipeline run. You should see that it was successful.



Toolchains / dev-zero-to-cloud-native-toolchain / dev-tekton-status-02cn /

dev-tekton-status-02cn Dashboard

PipelineRuns

Definitions

Worker

Triggers

Environment properties

Other settings

Trigger to EventListener mappings may need to be reconfigured due to changes in the Defi

Status: All Trigger: All

	#	Status	Name	Pipeline
☐	2	✓	pipelinerun-85dac128-7d9d-4dbe-...	pipeline

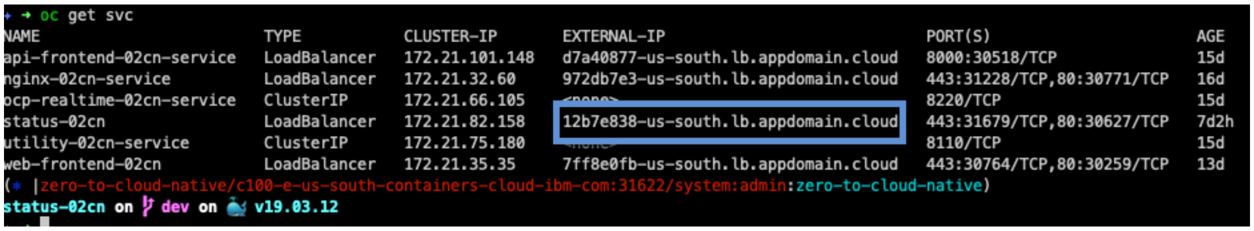
Items per page: 30 1 - 1 items

5 – Configure Cloud Internet Services

Next, we need to map that status application in Cloud Internet Services so we can expose the application to our domain.

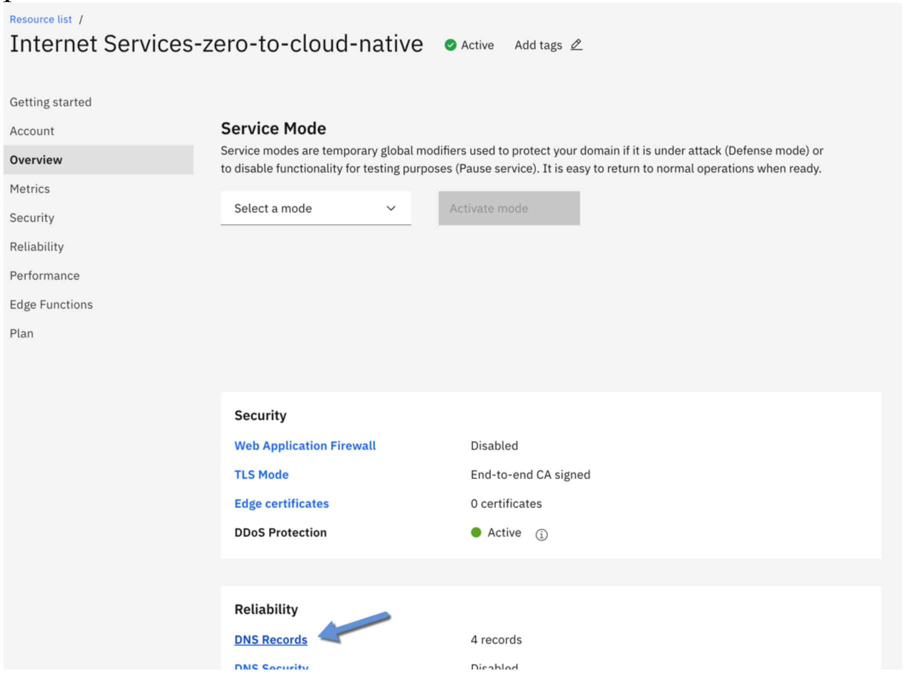
If you look in the deployment file for the status-02cn application, you will see that we create a load balancer service. We need to map the load balancer we just created to our Cloud Internet Services domain.

Start your iTerm2 terminal, log into your OpenShift Cluster, and make sure you are in the zero-to-cloud-native project. Next, run `oc get svc` to retrieve a list of services. Note the 'External-IP' for the status-02cn service.



```
+ → oc get svc
NAME                      TYPE           CLUSTER-IP      EXTERNAL-IP                             PORT(S)                      AGE
api-frontend-02cn-service LoadBalancer   172.21.101.148  d7a40877-us-south.lb.appdomain.cloud    8000:30518/TCP               15d
nginx-02cn-service        LoadBalancer   172.21.32.60    972db7e3-us-south.lb.appdomain.cloud    443:31228/TCP,80:30771/TCP   16d
ocp-realtime-02cn-service ClusterIP      172.21.66.105   <none>                                  8220/TCP                     15d
status-02cn               LoadBalancer   172.21.82.158   12b7e838-us-south.lb.appdomain.cloud    443:31679/TCP,80:30627/TCP   7d2h
utility-02cn-service      ClusterIP      172.21.75.180   <none>                                  8110/TCP                     15d
web-frontend-02cn         LoadBalancer   172.21.35.35    7ff8e0fb-us-south.lb.appdomain.cloud    443:30764/TCP,80:30259/TCP   13d
(* |zero-to-cloud-native/c100-e-us-south-containers-cloud-ibm-com:31622/system:admin:zero-to-cloud-native)
status-02cn on dev on v19.03.12
```

Going back to your browser, click on your Cloud Internet Services instance from the IBM Cloud portal and click on **DNS Records**.



Resource list /

Internet Services-zero-to-cloud-native Active Add tags

Service Mode

Service modes are temporary global modifiers used to protect your domain if it is under attack (Defense mode) or to disable functionality for testing purposes (Pause service). It is easy to return to normal operations when ready.

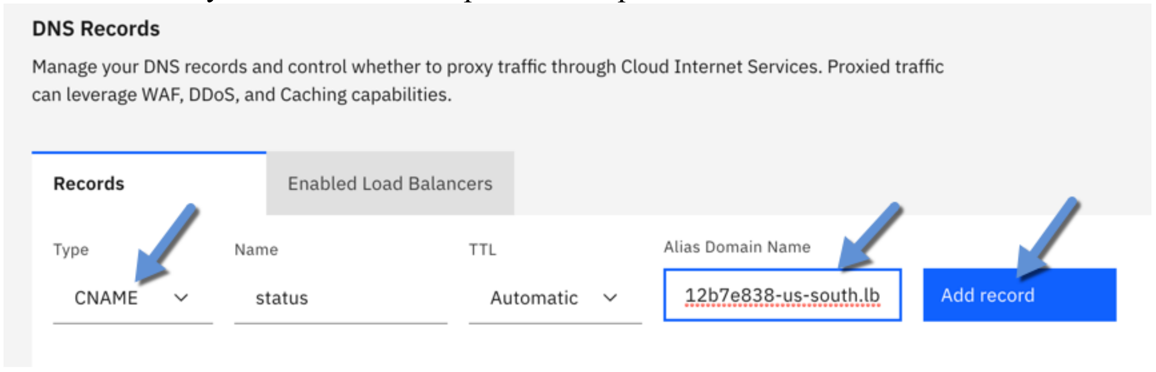
Security

Web Application Firewall	Disabled
TLS Mode	End-to-end CA signed
Edge certificates	0 certificates
DDoS Protection	● Active

Reliability

DNS Records	4 records
DNS Security	Disabled

As we did before, create a new **CNAME** record. Enter **status** as the name and paste the **load balancer URL** you retrieved in the previous step.



DNS Records

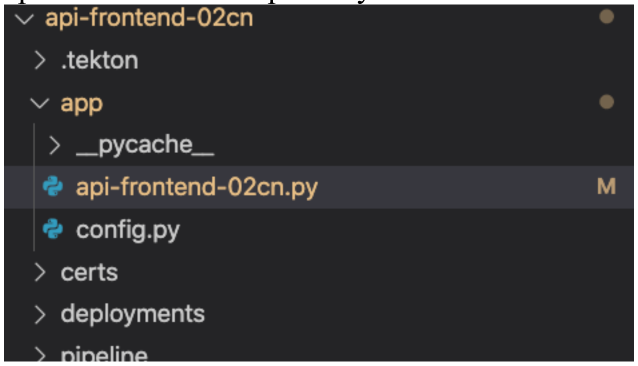
Manage your DNS records and control whether to proxy traffic through Cloud Internet Services. Proxied traffic can leverage WAF, DDoS, and Caching capabilities.

Type: CNAME
Name: status
TTL: Automatic
Alias Domain Name: 12b7e838-us-south.lb

6 – Customizing the API Microservice

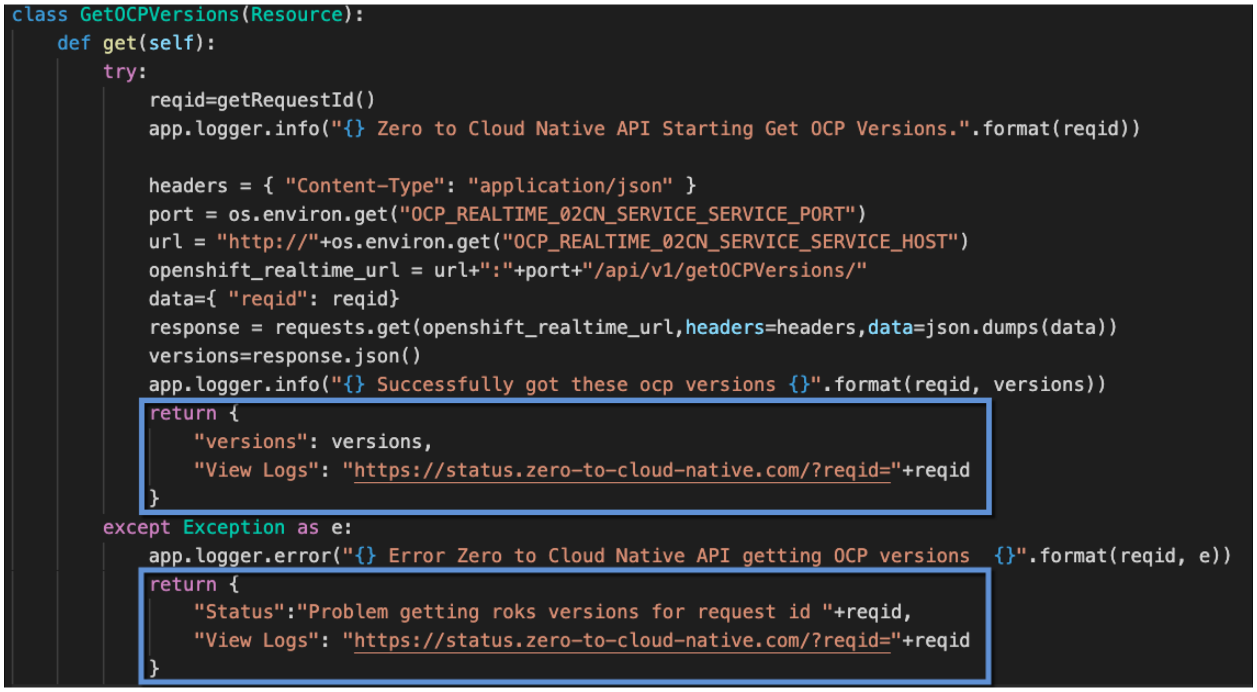
The last thing we need to do before we can test the status application is modify the API frontend application to return the status URL that a user can open to view log messages for their API requests.

From Visual Studio Code open the **api-frontend-02cn.py** file under in the app directory of the api-frontend-02cn repository.



Update all three APIs to return an additional field named View Logs.
In the **class GetOCPVersions** update the return blocks of code as follows:

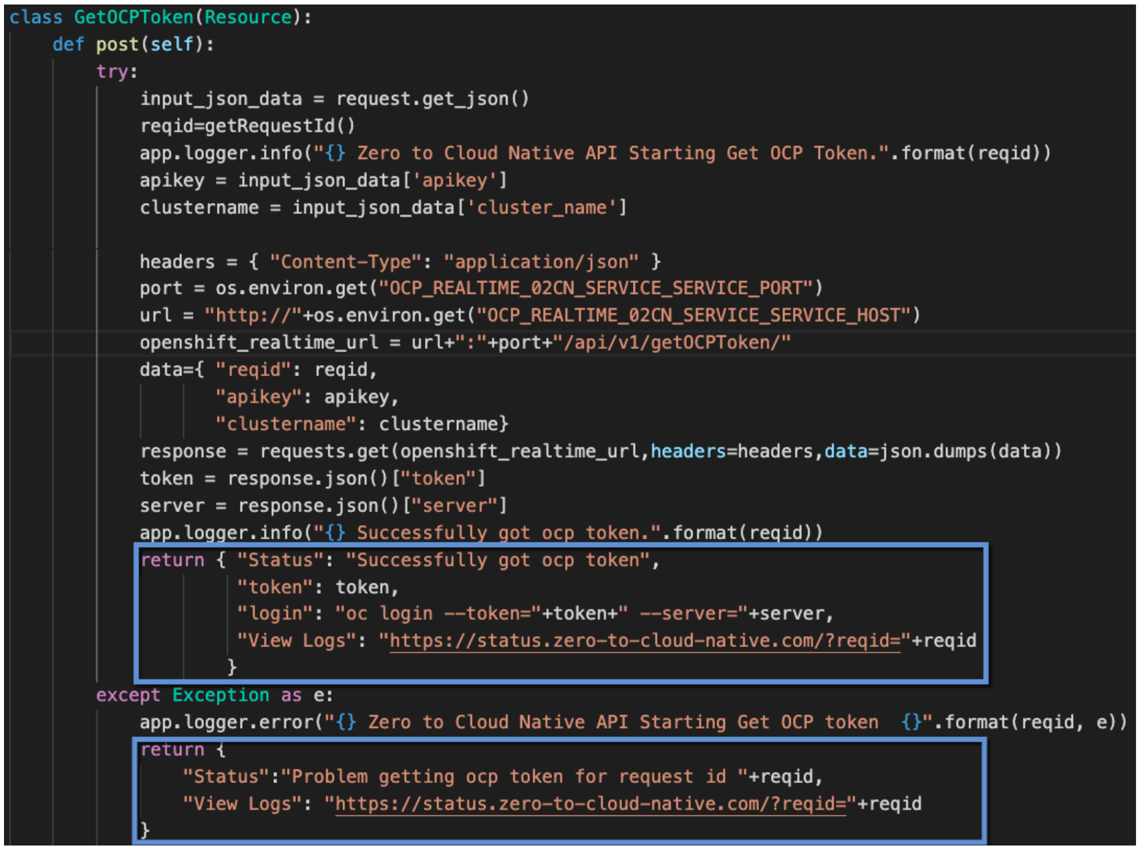
```
class GetOCPVersions(Resource):
    def get(self):
        try:
            reqid=getRequestId()
            app.logger.info("{} Zero to Cloud Native API Starting Get OCP
Versions.".format(reqid))

            headers = { "Content-Type": "application/json" }
            port = os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_PORT")
            url = "http://"+os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_HOST")
            openshift_realtime_url = url+":"+port+"/api/v1/getOCPVersions/"
            data={ "reqid": reqid}
            response = requests.get(openshift_realtime_url,headers=headers,data=json.dumps(data))
            versions=response.json()
            app.logger.info("{} Successfully got these ocp versions {}".format(reqid, versions))
            return {
                "versions": versions,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
        except Exception as e:
            app.logger.error("{} Error Zero to Cloud Native API getting OCP versions
{}".format(reqid, e))
            return {
                "Status":"Problem getting roks versions for request id "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
```



```
class GetOCPVersions(Resource):
    def get(self):
        try:
            reqid=getRequestId()
            app.logger.info("{} Zero to Cloud Native API Starting Get OCP Versions.".format(reqid))

            headers = { "Content-Type": "application/json" }
            port = os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_PORT")
            url = "http://"+os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_HOST")
            openshift_realtime_url = url+":"+port+"/api/v1/getOCPVersions/"
            data={ "reqid": reqid}
            response = requests.get(openshift_realtime_url,headers=headers,data=json.dumps(data))
            versions=response.json()
            app.logger.info("{} Successfully got these ocp versions {}".format(reqid, versions))
            return {
                "versions": versions,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
        except Exception as e:
            app.logger.error("{} Error Zero to Cloud Native API getting OCP versions  {}".format(reqid, e))
            return {
                "Status":"Problem getting roks versions for request id "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
```

GetOCPToken:

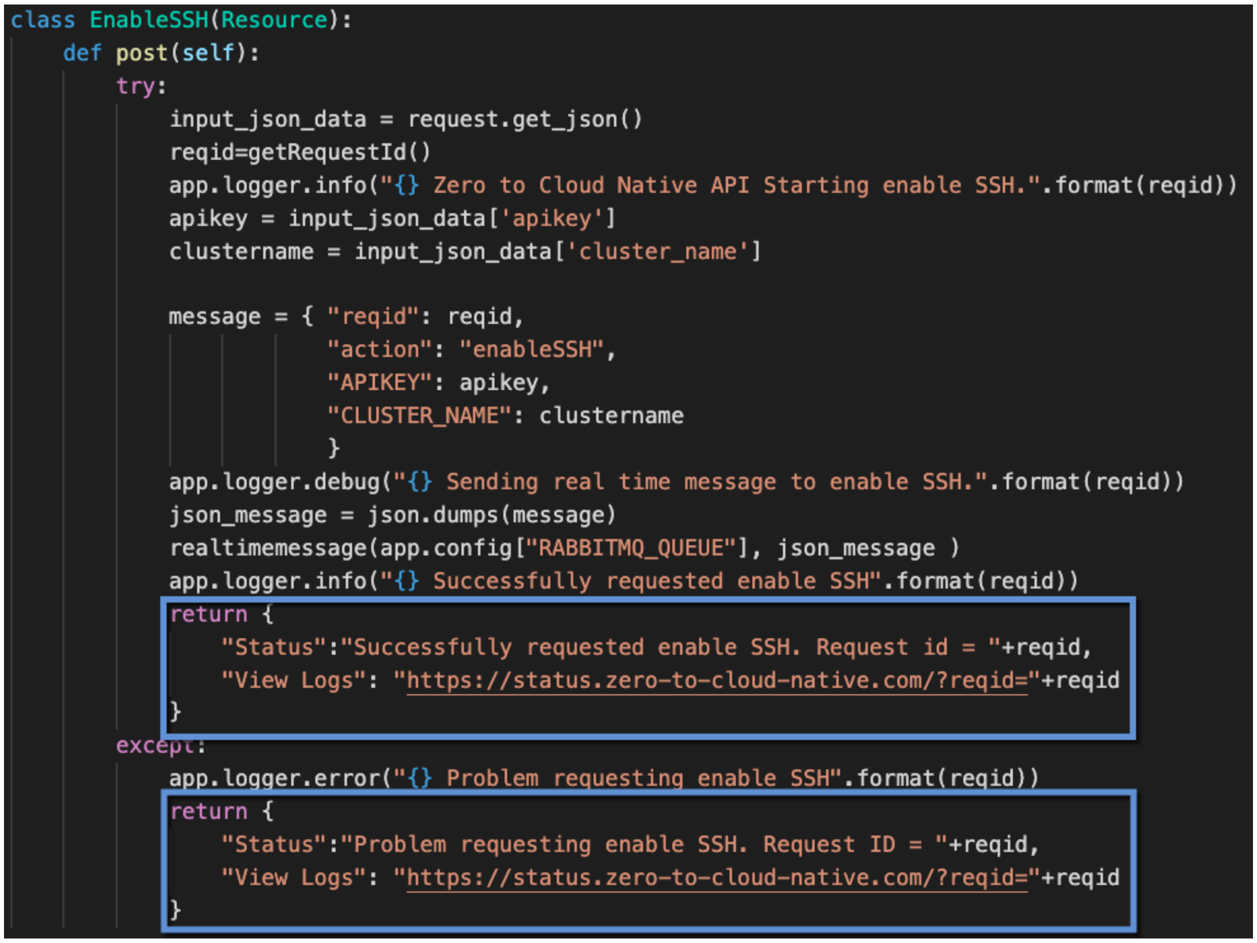
```
class GetOCPToken(Resource):
    def post(self):
        try:
            input_json_data = request.get_json()
            reqid=getRequestId()
            app.logger.info("{} Zero to Cloud Native API Starting Get OCP Token.".format(reqid))
            apikey = input_json_data['apikey']
            clustername = input_json_data['cluster_name']

            headers = { "Content-Type": "application/json" }
            port = os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_PORT")
            url = "http://"+os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_HOST")
            openshift_realtime_url = url+":"+port+"/api/v1/getOCPToken/"
            data={ "reqid": reqid,
                   "apikey": apikey,
                   "clustername": clustername}
            response = requests.get(openshift_realtime_url,headers=headers,data=json.dumps(data))
            token = response.json()["token"]
            server = response.json()["server"]
            app.logger.info("{} Successfully got ocp token.".format(reqid))
            return { "Status": "Successfully got ocp token",
                     "token": token,
                     "login": "oc login --token="+token+" --server="+server,
                     "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
                    }
        except Exception as e:
            app.logger.error("{} Zero to Cloud Native API Starting Get OCP token
{}".format(reqid, e))
            return {
                "Status":"Problem getting ocp token for request id "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
```



```
class GetOCPToken(Resource):
    def post(self):
        try:
            input_json_data = request.get_json()
            reqid=getRequestId()
            app.logger.info("{} Zero to Cloud Native API Starting Get OCP Token.".format(reqid))
            apikey = input_json_data['apikey']
            clustername = input_json_data['cluster_name']

            headers = { "Content-Type": "application/json" }
            port = os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_PORT")
            url = "http://"+os.environ.get("OCP_REALTIME_02CN_SERVICE_SERVICE_HOST")
            openshift_realtime_url = url+":"+port+"/api/v1/getOCPToken/"
            data={ "reqid": reqid,
                   "apikey": apikey,
                   "clustername": clustername}
            response = requests.get(openshift_realtime_url,headers=headers,data=json.dumps(data))
            token = response.json()["token"]
            server = response.json()["server"]
            app.logger.info("{} Successfully got ocp token.".format(reqid))
            return { "Status": "Successfully got ocp token",
                     "token": token,
                     "login": "oc login --token="+token+" --server="+server,
                     "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
                    }
        except Exception as e:
            app.logger.error("{} Zero to Cloud Native API Starting Get OCP token  {}".format(reqid, e))
            return {
                "Status":"Problem getting ocp token for request id "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
```

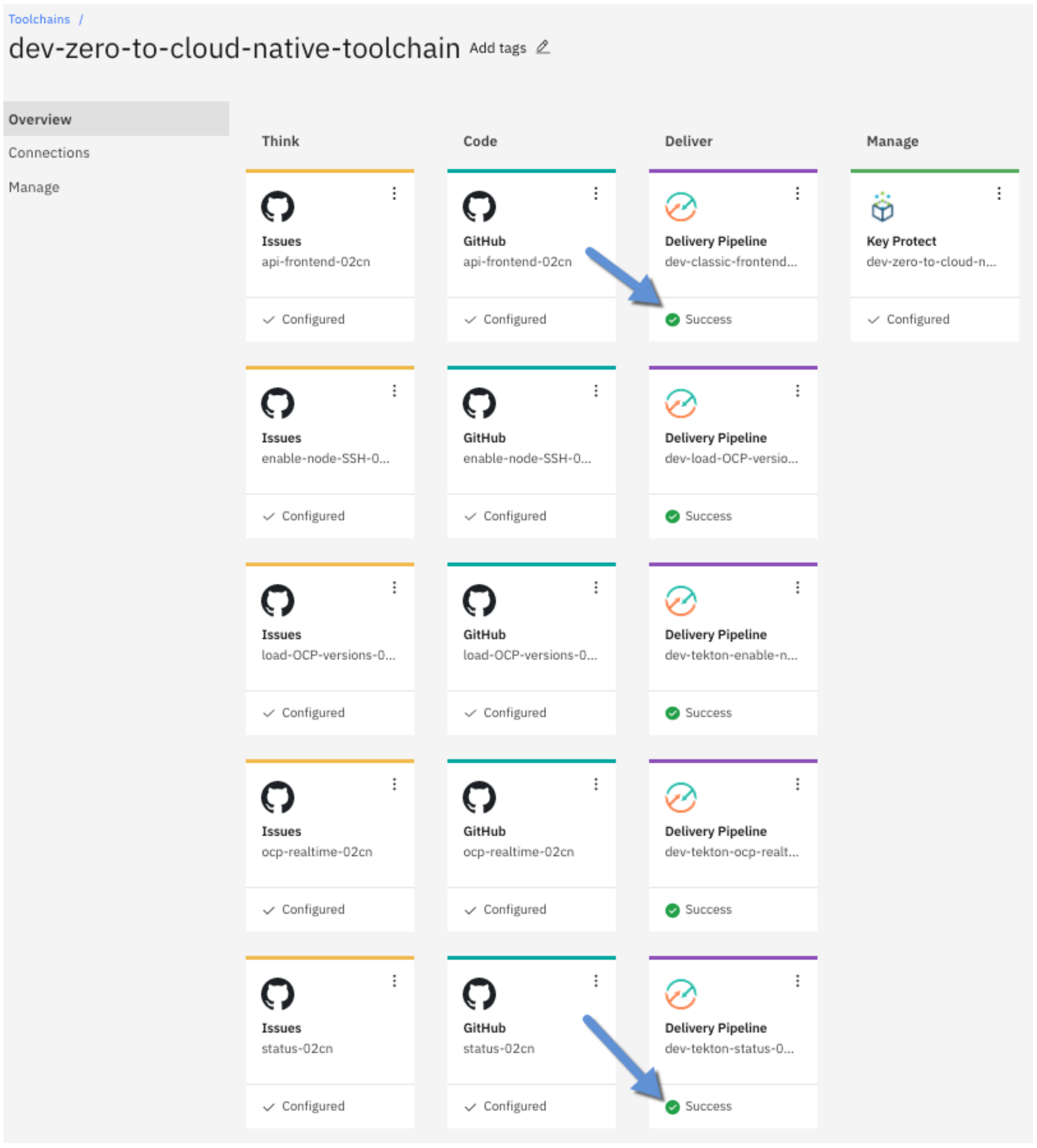
Enable SSH

```
class EnableSSH(Resource):
    def post(self):
        try:
            input_json_data = request.get_json()
            reqid=getRequestId()
            app.logger.info("{} Zero to Cloud Native API Starting enable SSH.".format(reqid))
            apikey = input_json_data['apikey']
            clustername = input_json_data['cluster_name']

            message = { "reqid": reqid,
                        "action": "enableSSH",
                        "APIKEY": apikey,
                        "CLUSTER_NAME": clustername
                        }
            app.logger.debug("{} Sending real time message to enable SSH.".format(reqid))
            json_message = json.dumps(message)
            realtimemessage(app.config["RABBITMQ_QUEUE"], json_message )
            app.logger.info("{} Successfully requested enable SSH".format(reqid))
            return {
                "Status":"Successfully requested enable SSH. Request id = "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
        except:
            app.logger.error("{} Problem requesting enable SSH".format(reqid))
            return {
                "Status":"Problem requesting enable SSH. Request ID = "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
```



```
class EnableSSH(Resource):
    def post(self):
        try:
            input_json_data = request.get_json()
            reqid=getRequestId()
            app.logger.info("{} Zero to Cloud Native API Starting enable SSH.".format(reqid))
            apikey = input_json_data['apikey']
            clustername = input_json_data['cluster_name']

            message = { "reqid": reqid,
                        "action": "enableSSH",
                        "APIKEY": apikey,
                        "CLUSTER_NAME": clustername
                        }
            app.logger.debug("{} Sending real time message to enable SSH.".format(reqid))
            json_message = json.dumps(message)
            realtimemessage(app.config["RABBITMQ_QUEUE"], json_message )
            app.logger.info("{} Successfully requested enable SSH".format(reqid))
            return {
                "Status":"Successfully requested enable SSH. Request id = "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
        except:
            app.logger.error("{} Problem requesting enable SSH".format(reqid))
            return {
                "Status":"Problem requesting enable SSH. Request ID = "+reqid,
                "View Logs": "https://status.zero-to-cloud-native.com/?reqid="+reqid
            }
```

After making those changes, **commit** both the **api frontend** and the **status** microservices to GitHub. After committing those changes, navigate to your toolchain and watch the pipelines run that will build and deploy the changes you just made.



Toolchains /

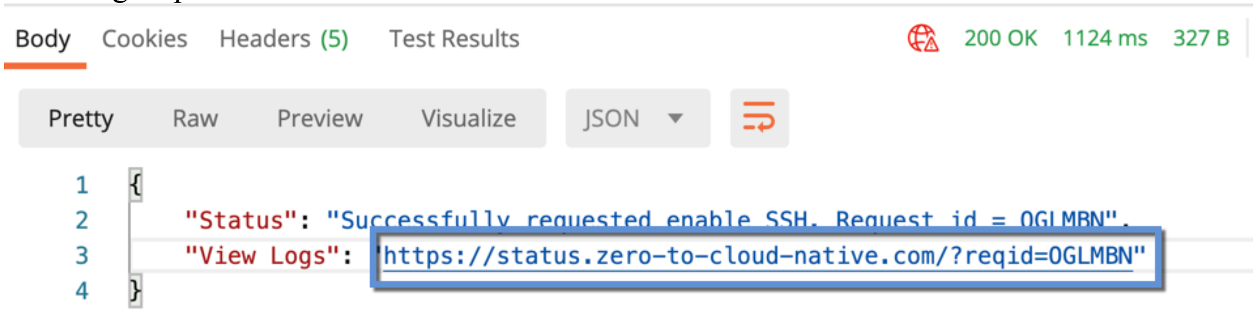
dev-zero-to-cloud-native-toolchain Add tags

Overview
Connections
Manage

Think	Code	Deliver	Manage
Issues api-frontend-02cn ✓ Configured	GitHub api-frontend-02cn ✓ Configured	Delivery Pipeline dev-classic-frontend... ● Success	Key Protect dev-zero-to-cloud-n... ✓ Configured
Issues enable-node-SSH-0... ✓ Configured	GitHub enable-node-SSH-0... ✓ Configured	Delivery Pipeline dev-load-OCP-versio... ● Success	
Issues load-OCP-versions-0... ✓ Configured	GitHub load-OCP-versions-0... ✓ Configured	Delivery Pipeline dev-tekton-enable-n... ● Success	
Issues ocp-realtime-02cn ✓ Configured	GitHub ocp-realtime-02cn ✓ Configured	Delivery Pipeline dev-tekton-ocp-realt... ● Success	
Issues status-02cn ✓ Configured	GitHub status-02cn ✓ Configured	Delivery Pipeline dev-tekton-status-0... ● Success	

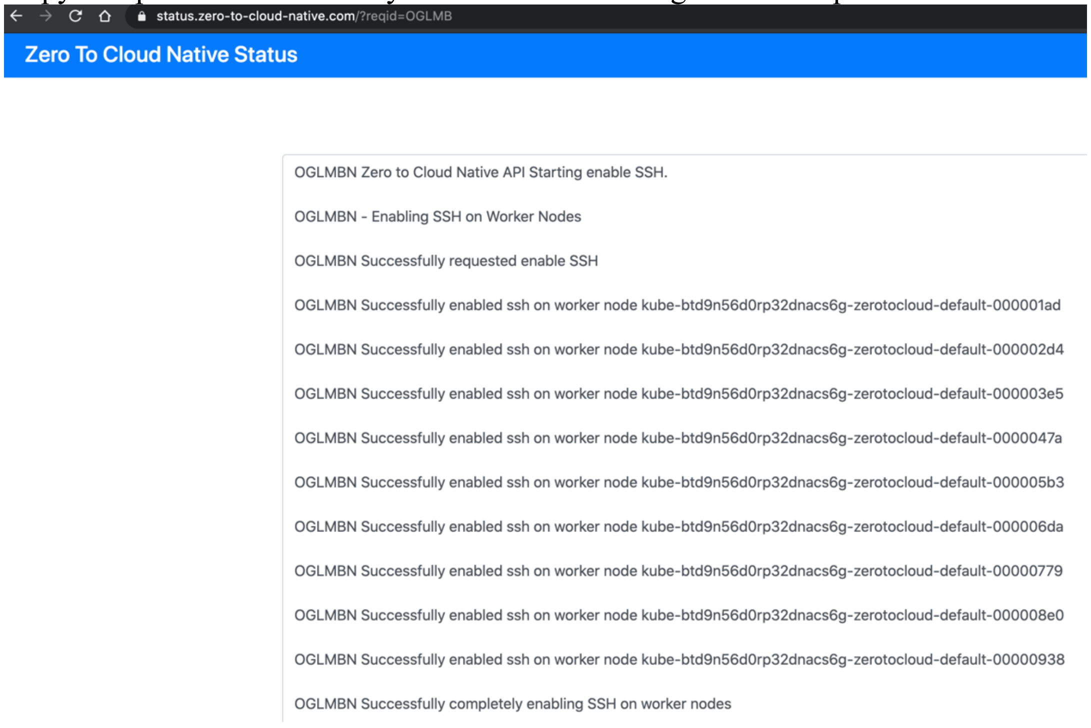
7 – Testing the Status Application

Pick your favorite API and run it.   In my case, I ran the Enable Node SSH API and got the following response:



```
{
    "Status": "Successfully requested enable SSH. Request id = OGLMBN",
    "View Logs": "https://status.zero-to-cloud-native.com/?reqid=OGLMBN"
}
```

Copy and paste the URL into your browser to see logs for the request.


status.zero-to-cloud-native.com/?reqid=OGLMB
Zero To Cloud Native Status

```
OGLMBN Zero to Cloud Native API Starting enable SSH.
OGLMBN - Enabling SSH on Worker Nodes
OGLMBN Successfully requested enable SSH
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000001ad
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000002d4
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000003e5
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-0000047a
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000005b3
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000006da
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-00000779
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-000008e0
OGLMBN Successfully enabled ssh on worker node kube-btd9n56d0rp32dnacs6g-zerotocloud-default-00000938
OGLMBN Successfully completely enabling SSH on worker nodes
```

Now, by simply using LogDNA, end users can see the status of their 'long' running API requests!

This completes the session on LogDNA. I hope these two sessions give you a good overview on why IBM has standardizing logging on IBM Cloud with LogDNA and you understand the type of things you can do with LogDNA. Check out our official documentation on Log Analysis with LogDNA to learn even more what LogDNA can do.

https://cloud.ibm.com/docs/Log-Analysis-with-LogDNA?topic=Log-Analysis-with-LogDNA-getting-started#getting-started