Open in app

Chris Graham

Follow About

Create a free K3s cluster in Oracle Cloud using the “Always Free Tier”

Chris Graham 3 days ago · 7 min read

When it comes to cloud hosting Kubernetes gets expensive quickly particularly for the hobbyist who wants to host applications or do some personal development.

Having stumbled across the Oracle Cloud free tier with their generous resource offerings “Always Free”. This seemed to be a perfect opportunity to get a capable always free cluster going.

Design goals:

- Utilise as much of the resources offered in free tier without using non-free resources
- Implementation should be as simple as possible to encourage other to enjoy the fruits of Kubernetes
- Have a reasonable amount of security without sacrificing usability
- Automated IaC bootstrapping
- Have a ready to deploy cluster including load balancers and Ingresses

Prerequisites:

1. An Oracle Cloud “Always Free Tier” subscription https://www.oracle.com/cloud/free/

2. Terraform installed https://learn.hashicorp.com/tutorials/terraform/install-cli#install-terraform

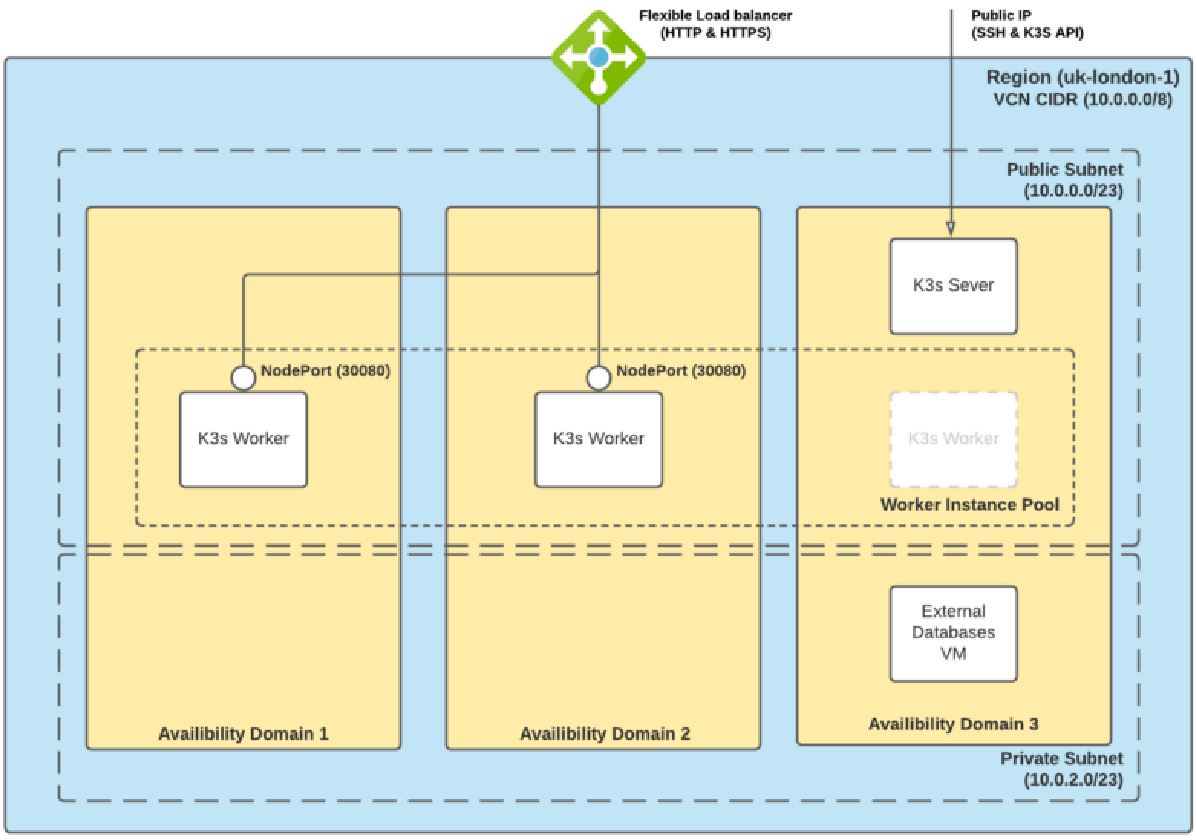
3. kubectl installed https://kubernetes.io/docs/tasks/tools/

4. Oracle Command Interface https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/cliinstall.htm

5. Either WSL2, MacOS or Linux terminal

Network Architecture Overview

In order to utilise the free tier resources as efficiently as possible the below design was settled on to meet the design goals. Below diagram indicates the final architecture once completing this guide.



Network & resource overview

The first amd64 instance is used as the K3s master and the second instance is used for stateful workloads such as SQL databases. The K3s worker nodes are set in an instance pool, this allows them to be fault tolerant across various availability domains.

Unfortunately as part of free tier we can only bootstrap the amd64 instances in availability domain three.

The free tier flexible load balancer is attached to a node port on each worker instance. This serves as the HTTP/HTTPS ingress endpoint from the load balancer to the nginx-ingress deployment.

An ephemeral IP is set for cluster administration (SSH & kubectl), this is locked down via whitelists.

Installing Dependancies

If you are running windows install WSL2 and install an Ubuntu instance.

Installation guide here https://ubuntu.com/wsl

Ubuntu Linux (Native or WSL)

```
# Install Terraform
sudo apt-get update && sudo apt-get install -y gnupg software-
properties-common curl jq git
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add
-
sudo apt-add-repository "deb [arch=amd64]
https://apt.releases.hashicorp.com $(lsb_release -cs) main"
sudo apt-get update && sudo apt-get install terraform

# Install OCI CLI
curl -LO https://raw.githubusercontent.com/oracle/oci-
cli/master/scripts/install/install.sh && chmod +x ./install.sh
sudo ./install.sh --install-dir /opt/oci --script-dir /opt/oci --
exec-dir /usr/local/bin/ --accept-all-defaults && rm ./install.sh
```

Authenticate against Oracle Cloud

Once the dependancies have been setup, we need to authenticate in order for terraform to use the authenticated details.

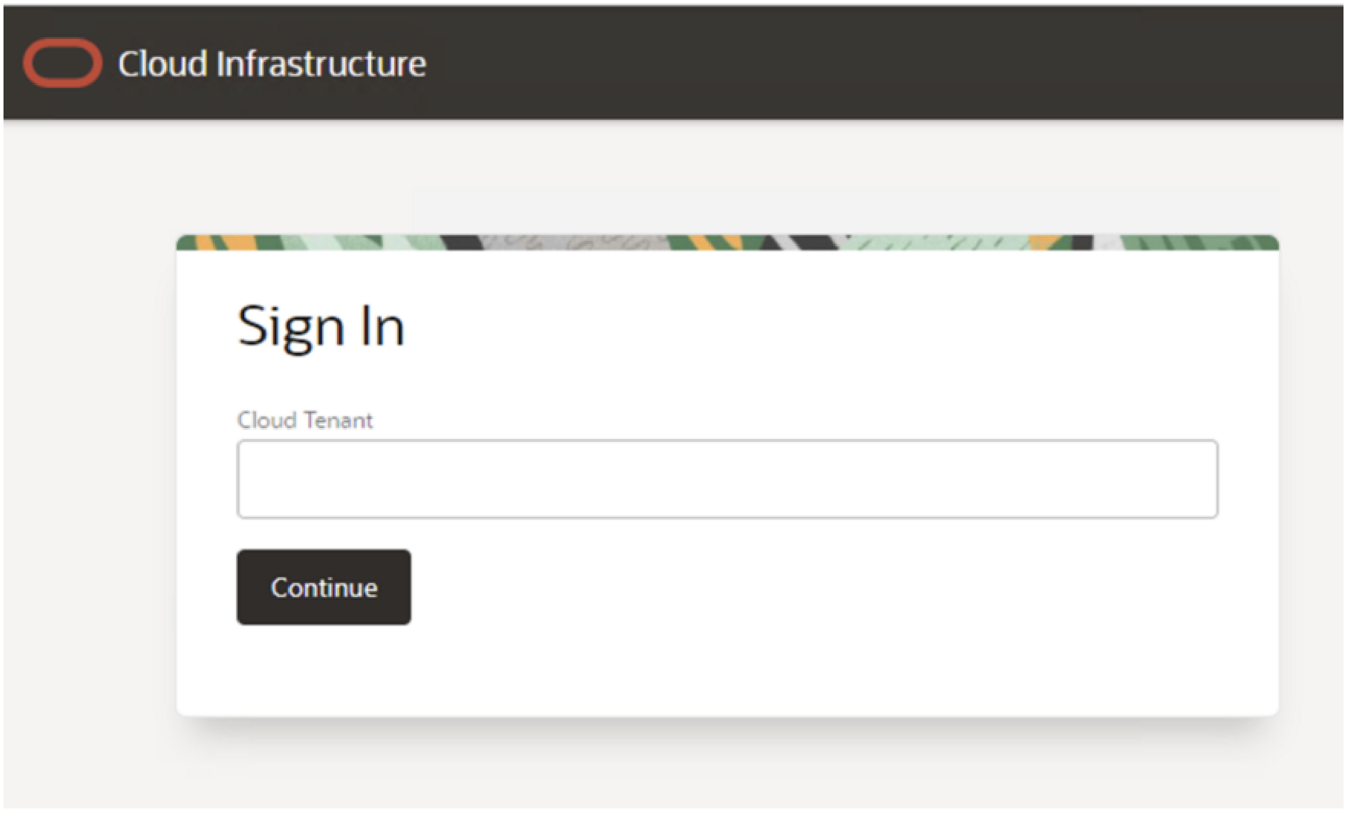
On a command prompt in your Ubuntu terminal run:

```
oci session --profile kubernetes authenticate
```

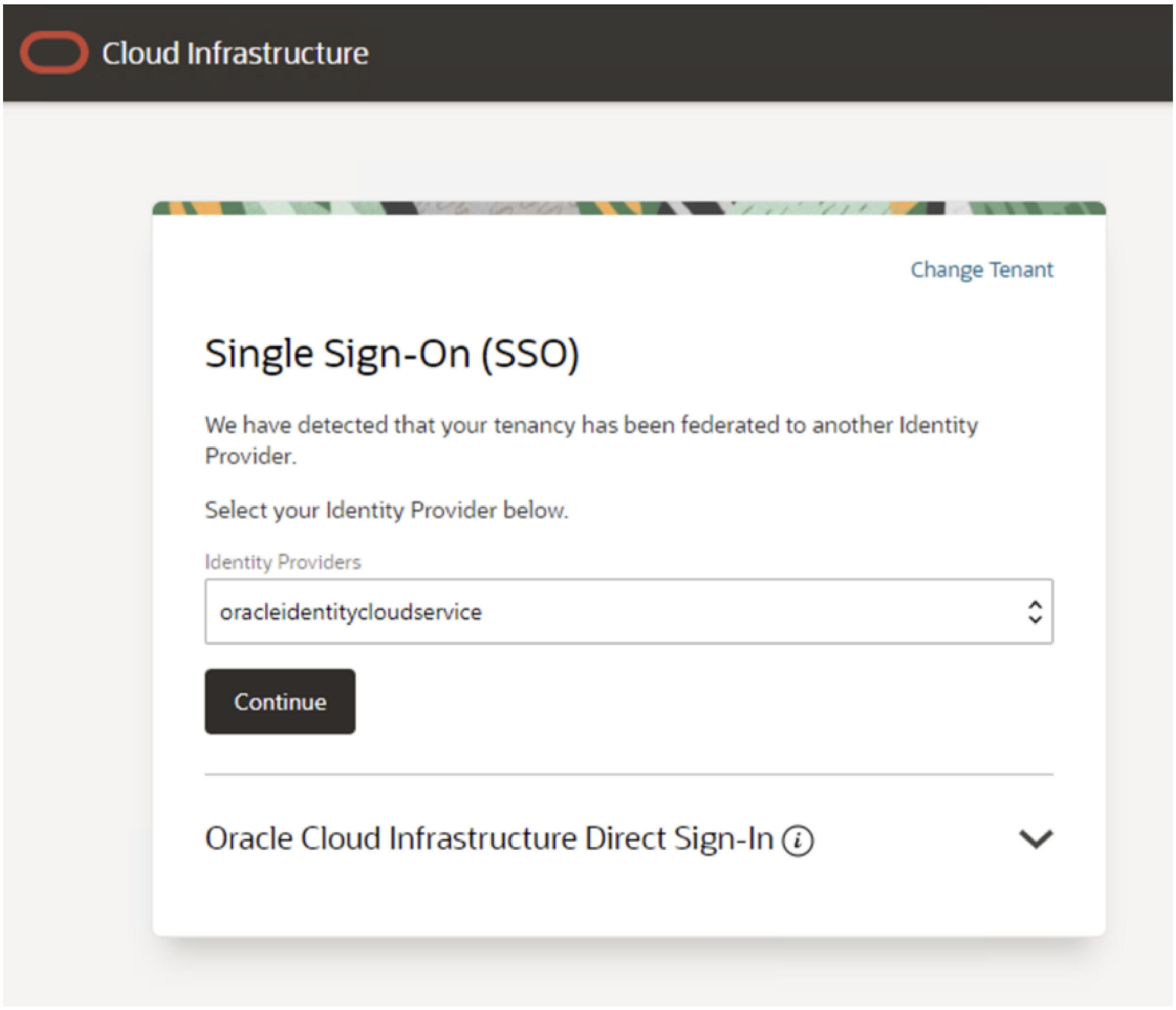
Select your home region you selected when signing up. In my case this is option 23 (uk-london-1).

```
Enter a region by index or name(e.g.
1: ap-chiyoda-1, 2: ap-chuncheon-1, 3: ap-hyderabad-1, 4: ap-
melbourne-1, 5: ap-mumbai-1,
6: ap-osaka-1, 7: ap-seoul-1, 8: ap-sydney-1, 9: ap-tokyo-1, 10: ca-
montreal-1,
11: ca-toronto-1, 12: eu-amsterdam-1, 13: eu-frankfurt-1, 14: eu-
zurich-1, 15: me-dubai-1,
16: me-jeddah-1, 17: sa-santiago-1, 18: sa-saopaulo-1, 19: sa-
vinhedo-1, 20: uk-cardiff-1,
21: uk-gov-cardiff-1, 22: uk-gov-london-1, 23: uk-london-1, 24: us-
ashburn-1, 25: us-gov-ashburn-1,
26: us-gov-chicago-1, 27: us-gov-phoenix-1, 28: us-langley-1, 29:
us-luke-1, 30: us-phoenix-1,
31: us-sanjose-1): 23
    Please switch to newly opened browser window to log in!
    You can also open the following URL in a web browser window to
continue:
https://login.uk-london-1.oraclecloud.com/v1/oauth2/authorize?
action=login&client_id=iaas_console&response_type=token+id_token&non
ce=xxxxxxx&scope=openid&public_key=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx&redirect_uri=http%3A%2F%2Flocalhost%3A
8181
```

In your browser pop-up sign in with your tenant name you signed up with.



You will then be directed to sign in via built in identity SSO:



Single Sign-On (SSO)

We have detected that your tenancy has been federated to another Identity Provider.

Select your Identity Provider below.

Identity Providers: oracleidentitycloudservice

Finally you will be directed to sign in with your username and password.

Note: Your may be required to re-authenticate should your session expire. You can renew your token by running the below again:

```
oci session --profile kubernetes authenticate
```

Configuring Terraform

Fetch the terraform and manifest files at https://github.com/GlorifiedTypist/k3s-oracle-cloud-free-tier

```
git clone https://github.com/GlorifiedTypist/k3s-oracle-cloud-free-
tier.git
```

There are two files in the directory that require configuration. The first file is the terraform.tfvars file. In this file, update the region variable in line with your home region:

terraform.auto.tfvars

```
# This must be your home region to take advantage of the free tier
region = "uk-london-1"
```

The second file is where the main configuration resides. Values that need to be reviewed or replaced as highlighted as bold.

main.tf

```
module "free-tier-k3s" {
  source = "./modules/free-tier-k3s"

  # General
  project_name   = "ftk3s"
  region         = var.region
  compartment_id = "ocid1.tenancy.oc1..aaaaaaaxxxxxxxxxxxxxxxxxyyyy"
  ssh_public_key = file("~/.ssh/id_rsa.pub")

  # Network
  whitelist_subnets = [
    "172.217.170.4/32",
    "10.0.0.0/8"
  ]

  vcn_subnet     = "10.0.0.0/16"
  private_subnet = "10.0.2.0/23"
  public_subnet  = "10.0.0.0/23"

  freetier_ad_list = [
    "gsEM:UK-LONDON-1-AD-3"
  ]
}
```

project_name: project name identifier, this can be anything to uniquely identify the infrastructure created by terraform

compartment_id: the value of the root compartment, this is the value of:

```
oci iam compartment list --config-file $HOME/.oci/config --profile
kubernetes --auth security_token | jq '.data[0].id'
```

ssh_public_key: public key used to connect to the instances via SSH. Ensure this file exists for the current user or generate a new one by running:

```
ssh-keygen
```

whitelist_subnets: SSH and K3S API whitelist source addresses. Here you can add your home static IP or a range of IPs in CIDR format. You can get your external IP by running:

```
curl ifconfig.co
```

If security is not your concern you can set this to [“0.0.0.0/0”, “10.0.0.0/16”]

Note: Must keep 10.0.0.0/16 here or the worker nodes will not be able to communicate with the k3s master.

freetier_ad_list: some regions such as uk-london-1 only allow amd64 nodes to be bootstrapped in a single availability domain. Add that domain ID here according to your region.

Running Terraform

Initialise terraform to pull the require dependancies:

```
terraform init
```

Validate the plan:

```
terraform plan
```

Finally run the terraform, not this takes around 10–12 mins to create all the infrastructure. Answer ‘yes’ to execute the plan:

Note: if you get an error ‘Error: 401-NotAuthenticated’, then re-authenticate as per the first steps in this guide.

```
terraform apply
```

Once completed, you will have several outputs. Take note of these we will be using them later.

```
k3s-api-ip = "x.x.x.x"
loadbalaner-ip = "http://y.y.y.y"
sqlpassword = "xxxxxxxxxxxxxxx"
```

But first test the cluster. Terraform created a Kubernetes config at ~/.kube/k3s so as not to overwrite any other cluster contexts you may have.

To reference the new cluster in the current terminal run:

```
export KUBECONFIG=~/.kube/k3s
```

Get a list of nodes:

```
$ kubectl get nodes
NAME                       STATUS   ROLES                  AGE
VERSION
inst-al0vh-ftk3s-worker   Ready    <none>                 24h
v1.21.2+k3s1
inst-ycted-ftk3s-worker   Ready    <none>                 24h
v1.21.2+k3s1
server                     Ready    control-plane,master   24h
v1.21.2+k3s1
```

Get a list of node resources consumed:

```
$ kubectl top nodes
NAME                       CPU(cores)   CPU%   MEMORY(bytes)
MEMORY%
inst-al0vh-ftk3s-worker    29m          1%     1367Mi           12%
inst-ycted-ftk3s-worker    21m          1%     1379Mi           12%
server                     73m          3%     461Mi            67%
```

Running a Test Deployment

A very simple nginx deployment is in the manifests folder. From the root of the directory run:

```
$ kubectl apply -f manifests/nginx-test.yaml
deployment.apps/test-nginx created
service/test-nginx created
ingress.extensions/test-nginx created
```

Now as we will not setup external DNS to point to the load balancer IP in this guide. We can simple test by setting the host header as defined in the nginx ingress deployment component.

manifests/nginx-test.yaml

```
...
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
  labels:
    app: test-nginx
  name: test-nginx
spec:
  rules:
    - host: ocft.nginx-test.doesntexist
      http:
        paths:
          - backend:
              serviceName: test-nginx
              servicePort: 80
...
```

Using the values of the terraform output:

```
k3s-api-ip = "x.x.x.x"
loadbalaner-ip = "http://y.y.y.y"
sqlpassword = "xxxxxxxxxxxxxxx"
```

Run curl with the above host and IP. This will return the nginx welcome page:

```
$ curl -H "Host: ocft.nginx-test.doesntexist" http://y.y.y.y
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully
installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Destroying the Cluster

As the cluster is automated via IaC. You can destroy and recreate whenever you need a new stack. To teardown all the resources created by terraform run:

```
terraform destroy
```

Hopefully this guide may have encourage at least one or two people to realise the benefits of embracing Kubernetes for their next project.