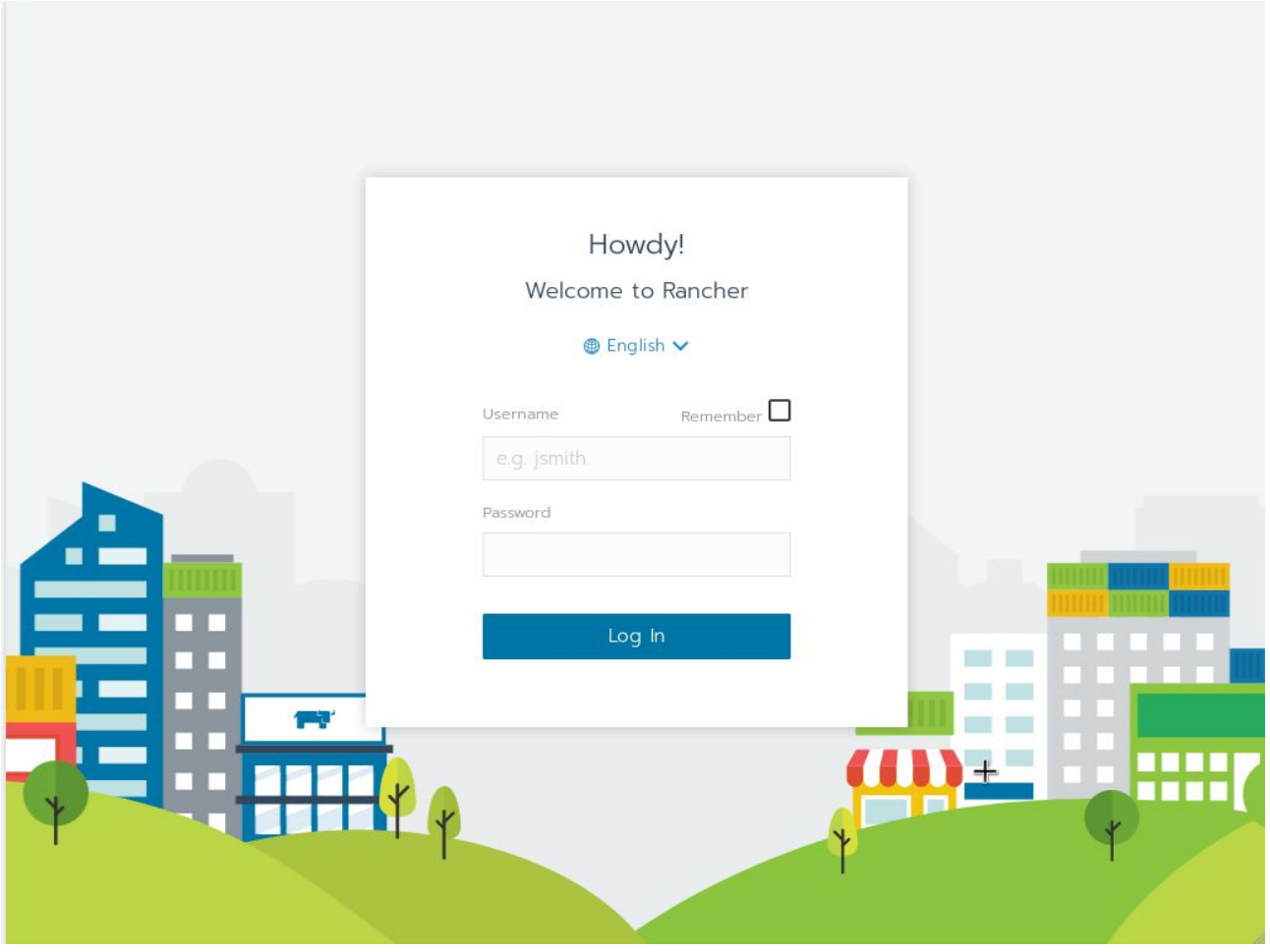
Kubernetes by examples (Rancher UI)

Instructor: Krerk Piromsopa, Ph.D.
Computer Engineering, Chulalongkorn University

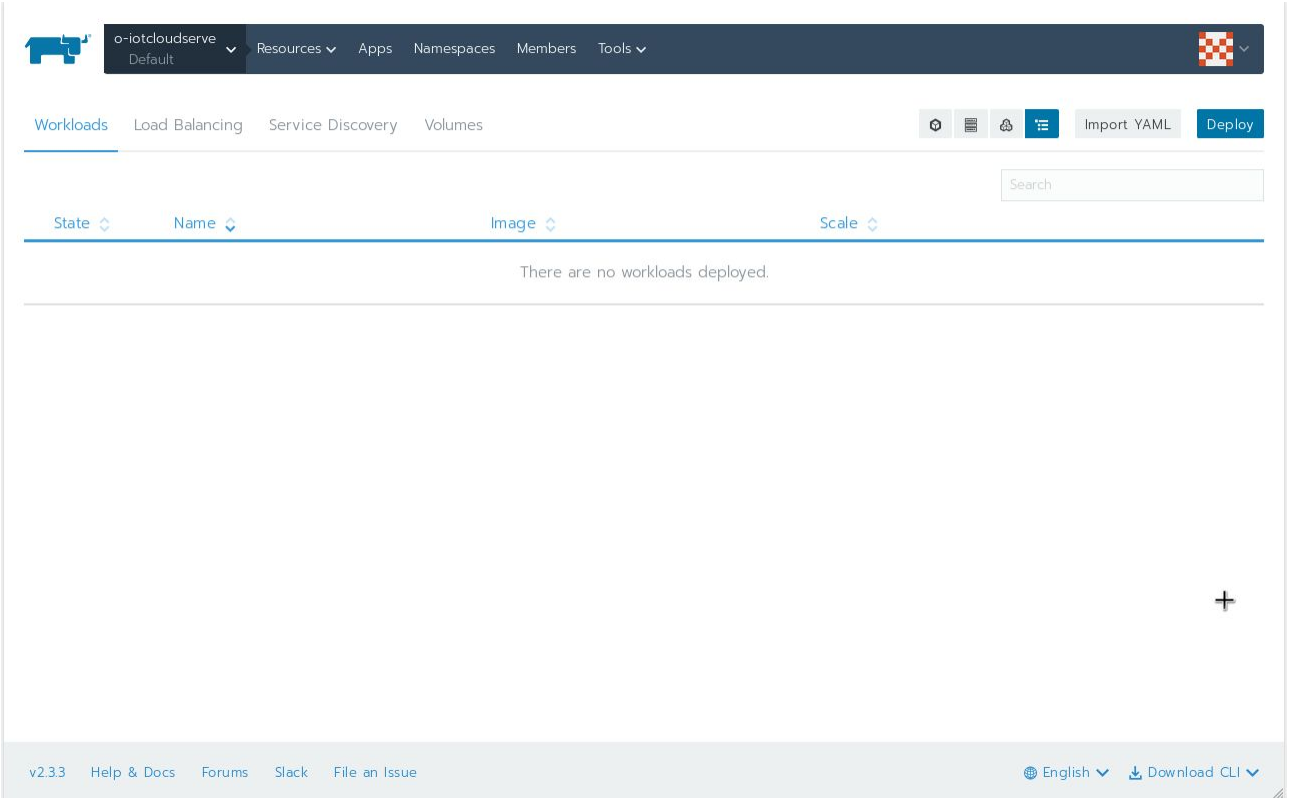
Overview

In this activity, we will learn to deploy (simple) cloud-native applications on kubernetes cluster using Rancher UI. Before we start, please first make sure that you can login to our Rancher with the given username and password. Our system is located at https://202.28.193.103/. (You may obtain the login information from the assistant.)



Exercises

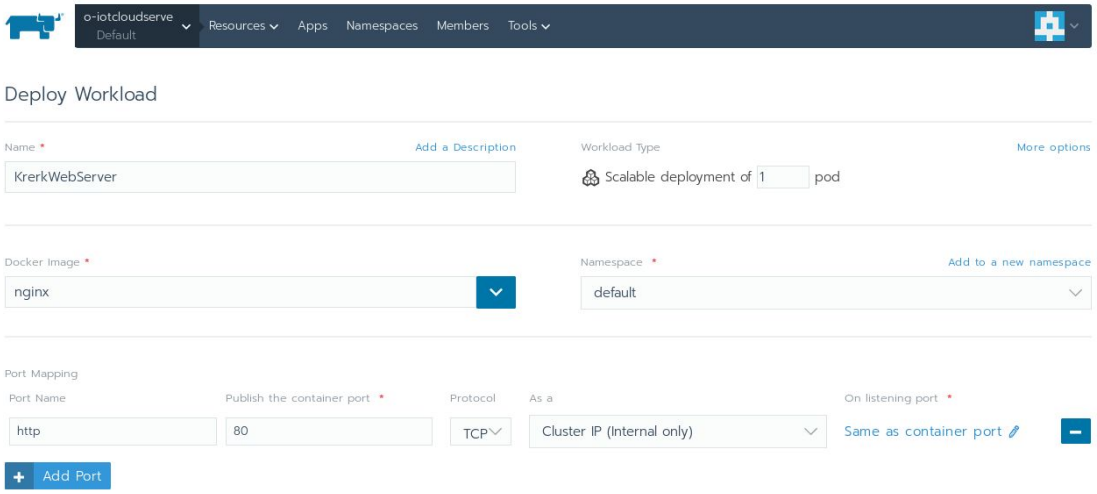
1. Simple stateless web application.
    a. We will first create a deployment (pod) using the Rancher UI.
    Select the default project by navigating to **Global > iotcloudserve > default.** You should now see the resources (workloads).



Deploy a new workload **nginx** by selecting ”Deploy”. Since we share a single namespace, prefix the name of the workload with something unique to you (e.g. your name). In this case, the workload name is KrerkWebServer. For the Docker image, put nginx. This is a standard HTTP service, se we have to public port http (80) as a ClusterIP. Here is the details information of your workload.

Name: Your workload name (Prefix with your name)
Wordload Type: Scalable deployment pod
Docker Image: nginx
Port Mapping:

- **Name:** Http
  Container Port: 80
  Protocol: TCP
  As : Cluster IP (Internal only)

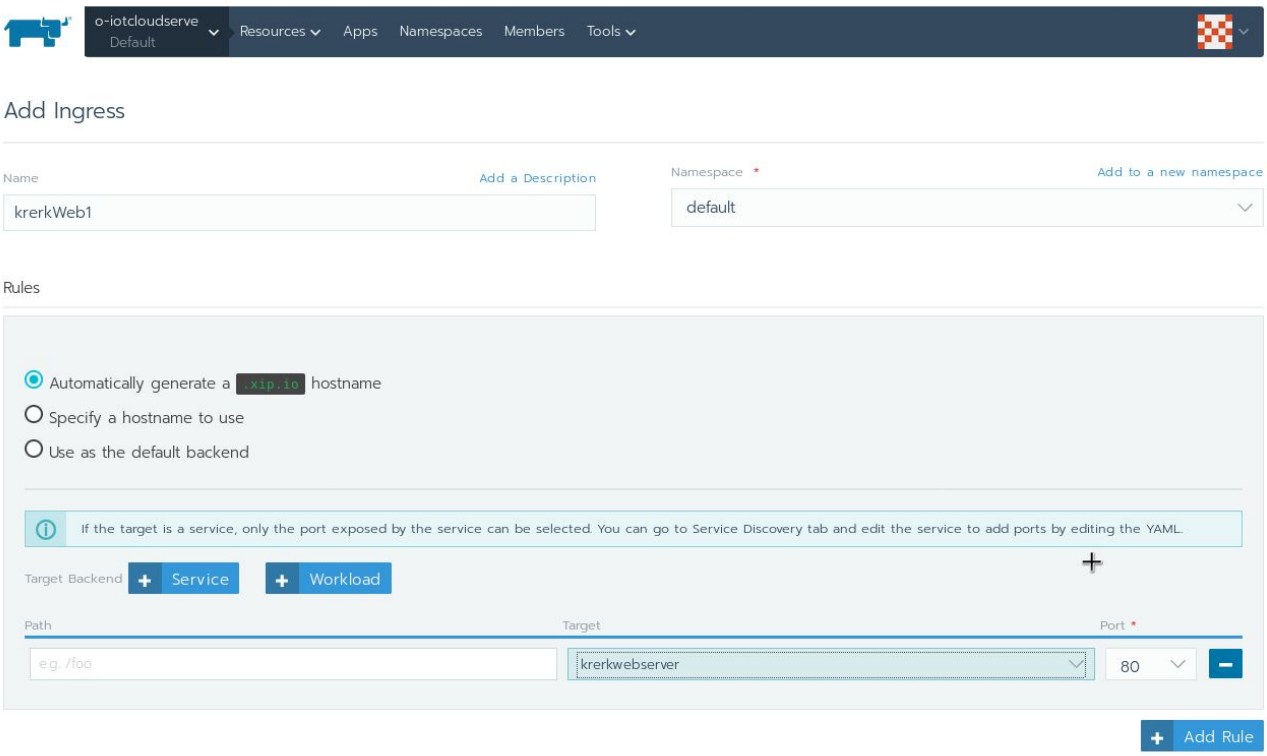


After launching, you will see that the pod has been created. Please note that Rancher will automatically expose the deployment as a service for you. To see the exposed service, navigate to **Service Discovery**. (If you deploy the workload with kubectl command line, you may have to manually expose the service).

b. To make the service accessible from the Internet, we can either (1) map the service port to the host public IP or (2) create an ingress load balancer[1] for you instance. In this case, we will create an ingress load balancer. Since, we do not have a pre registered DNS hostname available, we will use the free xip.io[2] service for the free dns. To create an ingress load balancer instance, navigate to **Load Balancing > Add Ingress**.

[1] Ingress load balancer is a kind of reverse proxy with virtual host service. It allows an IP address to be shared similar to named virtual host.

[2] Xip.io is a free public service that provides a wildcard DNS by resolving a hostname to an IP address embedded in the host. For example, **mysite.10.34.0.1.xip.ip** will be resolved to **10.34.0.1**. For more information, visit http://xip.io/.



Enter the name of you ingress. (Once again, prefix it with your name.) Now, use the For rules, use “Automatically generate a xip.io hostname”. Since we are exposing a predefined service, add a target for “Service”. Map the target to your registered service with port 80 (for our nginx). Note that it may take several minutes for the ingress controller to create the reverse proxy. You may now connect to the service with the registered host name. Here is the details information of your ingress configuration.

Name: A unique name for your ingress
Rules: Automatically generate a xip.io hostname
(You may want to delete the workload first.)
Services:

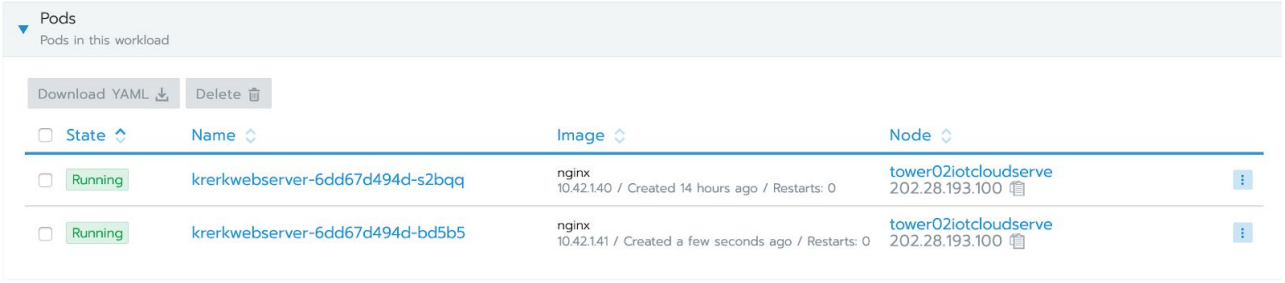
- **Path:** [leave empty]
  Target: [your service name]
  Port: 80

Once the service is ready, the record will be shown.

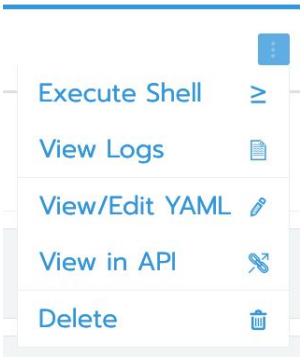
From the example, you can now navigate to http://krerkweb.default.202.28.193.100.xip.io/

c. Now, let’s scale our nginx service to 2 pods. At the workload tab, select our workload. In the details, there should be an entry name Config Scale. Change it to 2.

Let's wait until the new pod is created. You should see two pods now.



d. To find out which pod we are accessing, let's make a little modification to our pods. At the right-hand side of the pod, click the command button to show the menu. Now select **Execute Shell.**



Execute the following commands to each one.

```
# On pod 1
$ echo pod 1 > /usr/share/nginx/html/index.html

# On pod 2
$ echo pod 2 > /usr/share/nginx/html/index.html
```

Now if we go back to our browser (http://krerkweb.default.202.28.193.100.xip.io/) and try to refresh it several times, we will see that we are randomly redirected to different pods.

```
Note that any change to each pod are stateless. Once the pod
is recreated, it will be gone.
```

2. Stateful web application. In order to create a stateful application, we have to create a persistent volume. There are several ways to create a volume. The easiest way is to create it though RANCHER workload deployment. We will deploy a simple web-based file manager (created by myself). I have prepared the container image for you. To learn more about this app, visit https://gitlab.com/krerk/nuol-demo/. For now, let's deploy our application with the following details.

   Name: Your workload name (Prefix with your name)
   Workload Type: Scalable deployment pod
   Docker Image: registry.gitlab.com/krerk/nuol-demo
   Port Mapping:
   - **Name:** Http
     Container Port: 80
     Protocol: TCP
     As : Cluster IP (Internal only)

   Volumes:
   - Add a new persistent volume (claim)
     Name: [Your volume name]
     Use Storage Class: default
     Capacity: 0.1 Gb
     Access Modes:
     - Many Nodes Read-Write
   - **Mount Point:** /var/www/data

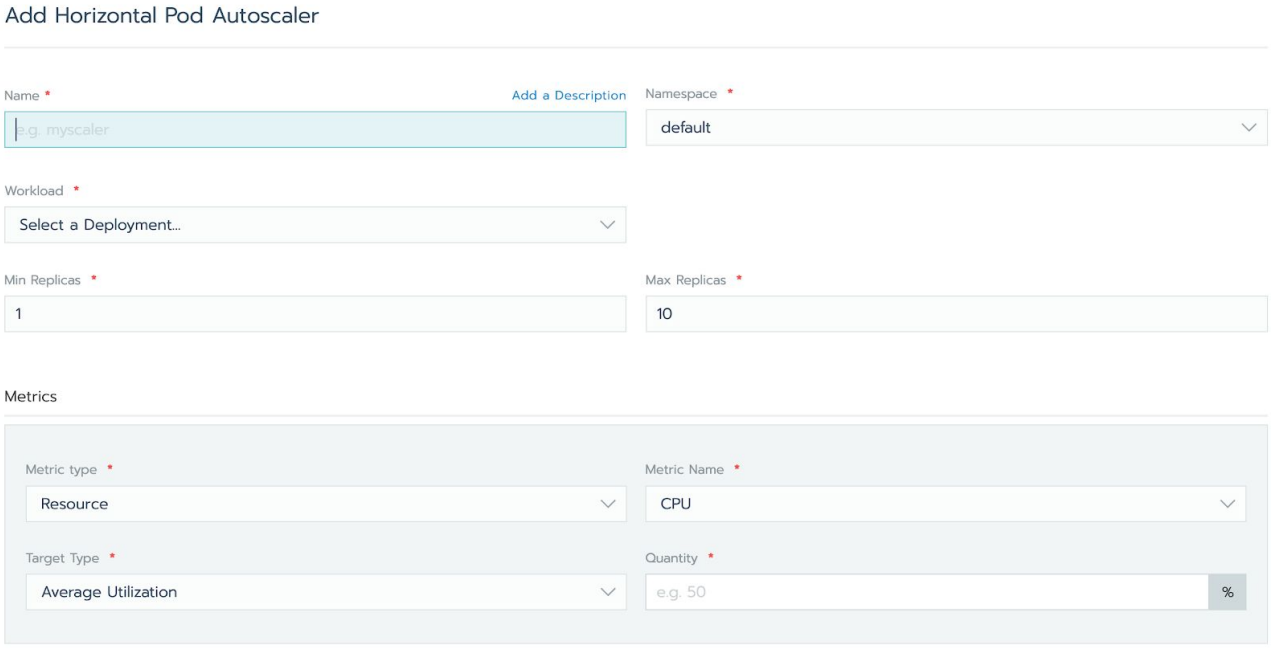
   Now create an ingress service for you services. (Don't forget to prefix the ingress name with your name.) Once finished, we can now access our service.

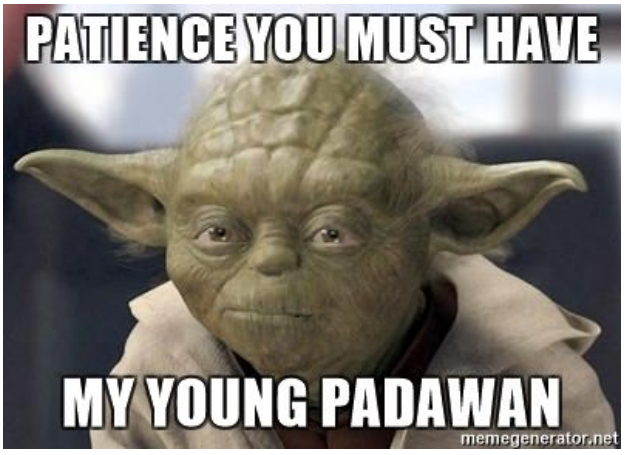
   To see that it is a stateful application, try decreasing the replica set to 0. When the service is back, the data will still persist. This is because the pod will mount the same persistence volume back.

3. Horizontal Pod Autoscaling. In this exercise, we will learn to dynamically scale our kubernetes based on utilization. By default, Kubernetes supports resource metric (CPU and Memory). However, you use custom metric. Nonetheless, this is beyond the scope of this exercise.
   a. In our Rancher UI, navigate to **Resources > HPA.** Select **Add HPA** with the following details.
      Name: [Name of your HPA]
      Workload: [Name of your workload]
      Min Replicas: 1
      Max Replicas: 5
      Metrics:
      - **Metric type:** Resource

Metric Name: CPU
Target Type: Average Value
Quantity: 4 milli CPU[3]

b. Now that you have set the metric. Try to trigger the scale by refreshing several times for about 15 seconds.





[3] In reality, this number should be 600 mCPU or more.