Advanced Features

We now have a 'real life' application deployed and running. We have added the necessary pieces to consider this 'production ready'.

In this section we will cover

- Autoscaling
- Advanced Deployments

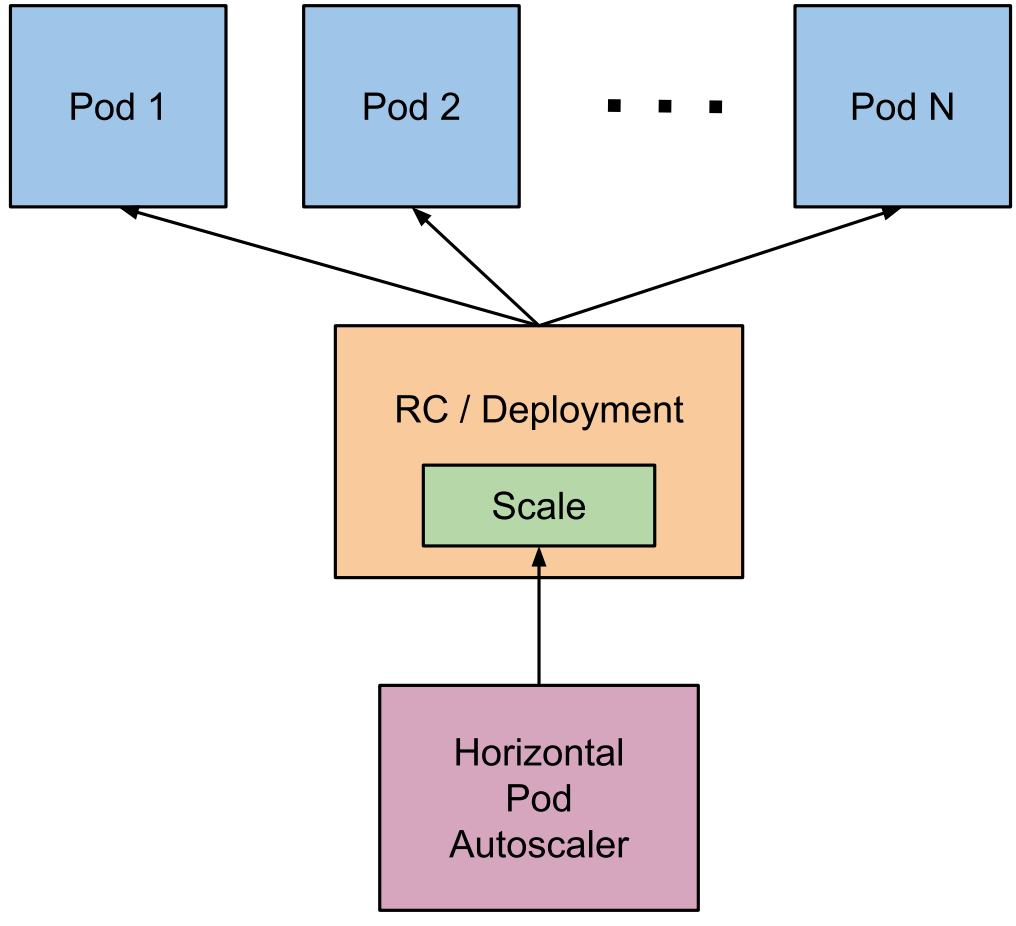
Auto Scaling

As we have seen in the previous section scaling our applications is very simple. `kubectl scale deployment/k8s-real-demo replicas=5`. However, ideally this would not be a manual action.

Kubernetes supports this via the `HorizontalPodAutoscaler` resource.

HorizontalPodAutoscaler (HPA)

- Periodically fetches metrics (default 30 seconds)
- Compares to user specified target value.
- Adjusts the number of replicas (pods) of a Deployment if needed.
- CPU usage is built in.
- Fetched from Heapster.
- Can also read Prometheus (for custom metrics)


Pod 1
Pod 2
Pod N
RC / Deployment
Scale
Horizontal
Pod
Autoscaler

Adding an HPA

We can add an HPA to our existing Deployment

./resources/hpa.yaml

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: k8s-real-demo-hpa
spec:
  scaleTargetRef:
    kind: Deployment
    name: k8s-real-demo
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 80
```

We can create this resource on the cluster just as we have done with the others:

```
$ kubectl apply -f ./resources/hpa.yaml
```

And view the resource with `kubectl get` and `kubectl describe`

Testing the HPA

In order to test that the HPA works as expected we will need our application to consume > 80% CPU. Luckily our application has a very CPU intensive endpoint (`/mineBitcoin`).

Make a request (or several) to the endpoint `/mineBitcoin` (note this endpoint can take a query parameter `seconds`).

See if your Deployment scales successfully.

Advanced Deployments

Kubernetes offers a variety of ways to release an application.

The correct option depends on your specific requirements and use case.

In this section we will try out several options and look at some possible use cases.

Deployment Strategies

- **Recreate**: Terminate the old version then release a new one
- **Ramped**: Release a new version via a rolling update
- **Blue/Green**: Release a new version alongside the old version then switch traffic
- **Canary**: Release a new version to a subset of users, then proceed to a full rollout
- **A/B testing**: release a new version to a subset of users in a precise way (HTTP headers, cookie, weight, etc.).

Recreate deployment

First terminate (all instances of) the old version and then release the new one.

Add the below `strategy` section to your existing deployment.yaml and remove the `env` section

```
...
spec:
  replicas: 3
  strategy:
    type: Recreate
...
```

```
$ kubectl apply -f ./resources/deployment.yaml
```

Verify the deployment:

```
$ curl $EXTERNAL_IP:[NodePort]
Hello from Container Solutions.
I'm running version 2.1 on k8s-real-demo-5449767b94-hnk78
```

In the output we can see both the Pod ID as well as the version of the application.

To see the deployment in action, open a new terminal and run the following command in another terminal:

```
$ watch -n1 kubectl get po
```

or

```
$ kubectl get po -w
```

Then deploy the version 2 of the application:

```
$ kubectl set image deploy/k8s-real-demo k8s-real-demo=icrosby/k8s-re
```

Now test the second deployment progress.

(N.B. Since we are not removing the service, the NodePort will not change)

```
$ export SERVICE_URL=$EXTERNAL_IP:[NodePort]
$ while sleep 0.1; do curl $SERVICE_URL; done;
```

Blue/Green Deployment

Release a new version alongside the old version then switch traffic

Deploy the first application

```
$ kubectl apply -f ./resources/deployment.yaml
```

Test if the deployment was successful

```
$ curl $EXTERNAL_IP:[NodePort]
Hello from Container Solutions.
I'm running version 1.0 on k8s-real-demo-5449767b94-hnk78
```

To see the deployment in action, open a new terminal and run the following command:

```
$ kubectl get po -w
```

Then create a second deploy with version 2 of the application:

```
$ kubectl apply -f ./resources/deployment-v2.yaml
```

Side by side, 3 pods are running with version 2 but the service still send traffic to the first deployment.

Try manually test one of the new pods by port-forwarding it to your local environment.

Once your are ready, you can switch the traffic to the new version by patching the service to send traffic to all pods with label app: k8s-real-demo-v2:

```
$ kubectl patch service my-app -p \
'{"spec":{"selector":{"app: k8s-real-demo-v2}}}'
```

Alternatively you can use

```
$ kubectl edit service k8s-real-demo
```

Test if the second deployment was successful:

```
$ export SERVICE_URL=$EXTERNAL_IP:[NodePort]
$ while sleep 0.1; do curl $SERVICE_URL; done;
```

In case you need to rollback to the previous version:

```
$ kubectl patch service my-app -p \
'{"spec":{"selector":{"version":"v1.0.0"}}}'
```

If everything is working as expected, you can then delete the v1.0.0 deployment:

```
$ kubectl delete deploy my-app-v1
```

Clean Up

```
$ kubectl deployment k8s-real-demo
```

Next up Setting up an HA cluster