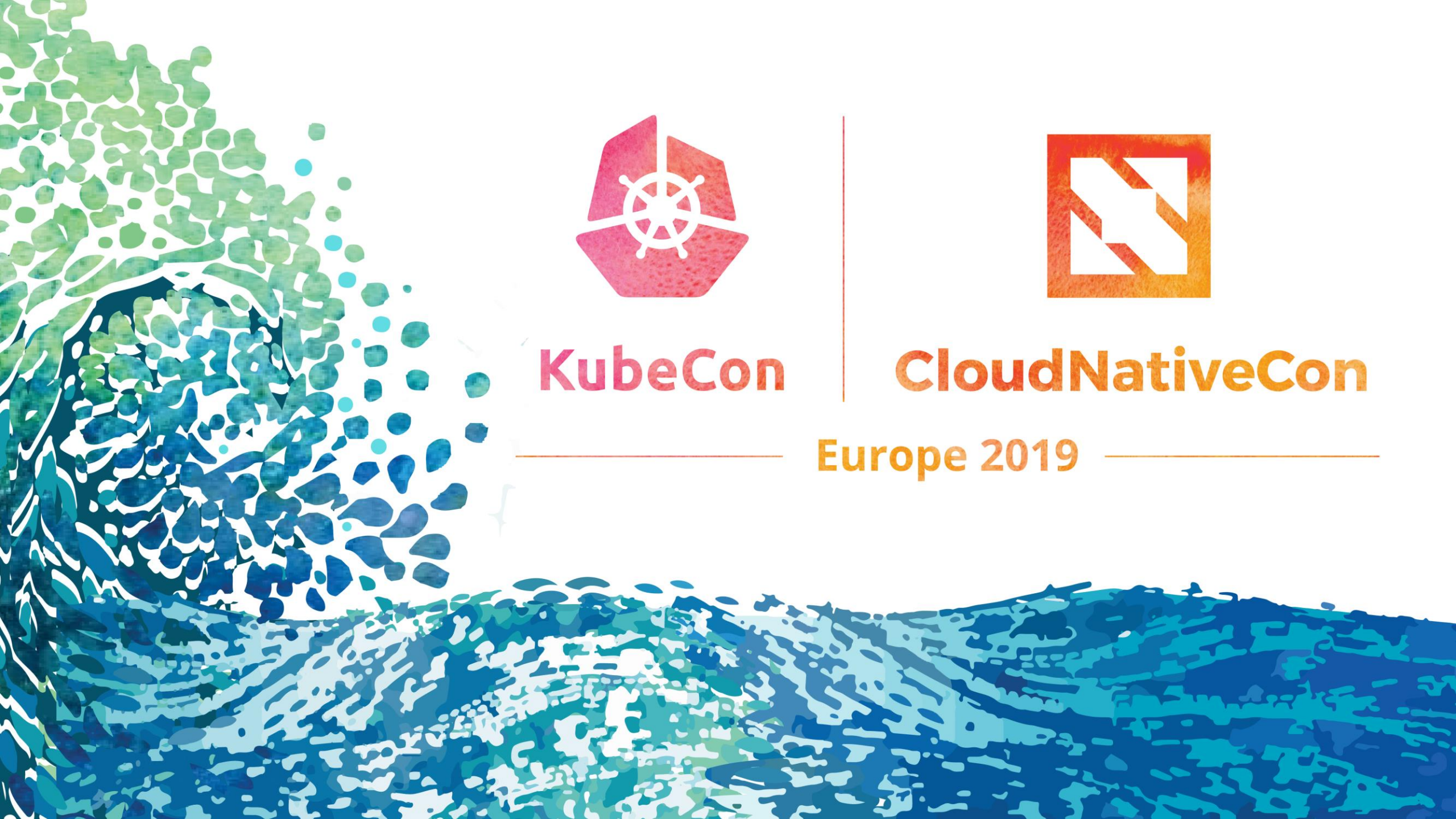

KubeCon
CloudNativeCon
Europe 2019

KubeCon | CloudNativeCon Europe 2019

A Day in the Life of a Cloud Native Developer

Randy Abernethy (RX-M, LLC)

Painless distributed application development

- This talk is designed to take (some of) the sting out of software engineering in a cloud native world
- We are going to do something small but useful with **9 CNCF open source projects in 90 minutes!**
  - Put down your phone or perish …

How this is going to work:

For x in [gRPC, Containerd, Harbor, K8s, Helm, Prometheus, Fluentd, Istio, Telepresence]:

- **I introduce the CNCF project – 45 seconds**
- **I overview the next step in the lab tutorial – 45 seconds**
- **You complete the lab tutorial step – 8 minutes**
- **I review what you [should | may] have [built | created | done]**

FAQ:

This is a crazy pace for a tutorial, no?

> Yes, yes it is

Why are you treating us so harshly when we just met you?

> I'm normally a nice guy but I want you to see how magical it is when all of these things work together and they only gave me 90 minutes, so you'll just have to power down a Red Bull or two and get fired up.

What if I need help?

> There are DevOps Demons wandering around, they can sort you (please don't give them any of the Red Bull).

Where's the code?

> https://github.com/RX-M/kubecon-eu-2019

Go here:

If you can type

https://github.com/RX-M/kubecon-eu-2019

https://bit.ly/30FXZOa

If you can not type but readily remember random sequences of 7 characters

Login!

- Everyone should have been handed **ssh creds** for a cloud instance running in AWS as they entered the room
  - If not see one of the DevOps Desperados
- Don’t listen to me, login!!!
- If you need help at anytime ping a helper or hit the chat to talk to people on the other side of the planet:
  - **https://zoom.us/j/758119466**

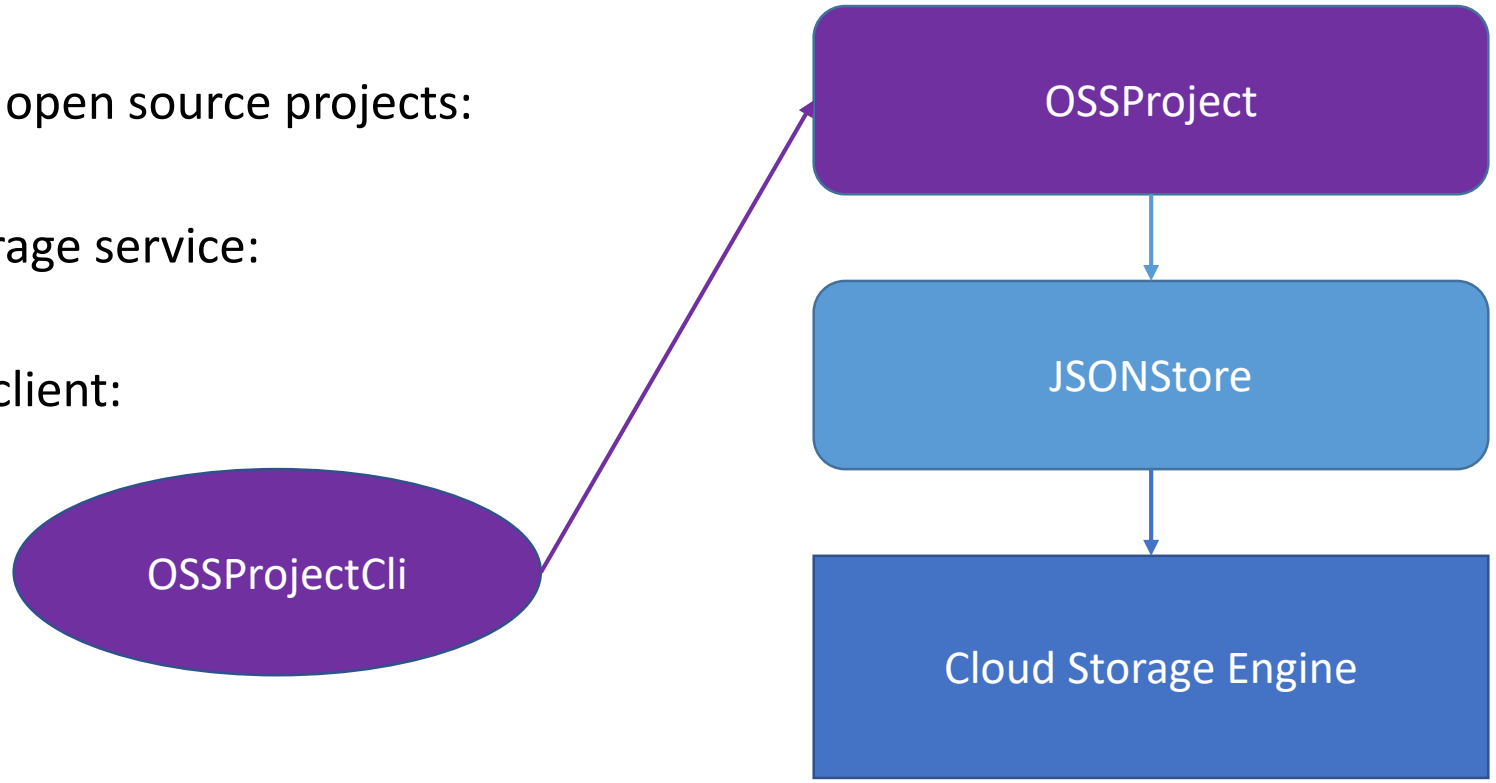
Cloud infra provided by amazon Web Services
Very cool folks over there…

Thanks Amazon!!

What are we Building?

- A gRPC service to help us track open source projects:
  - OSSProject
- Our service will consume a storage service:
  - JSONStore
- Our service will support a test client:
  - OSSProjectCli

We gotta keep it simple if we're going to get through all of this stuff!!



gRPC

gRPC is a recursive backronym standing for **gRPC Remote Procedure Call**

- An open source remote procedure call (RPC) system initially developed at Google
- Uses HTTP/2 for transport and Protocol Buffers for IDL and serialization
- Provides features such as:
  - Authentication
  - Bidirectional streaming and flow control
  - Blocking or nonblocking bindings
  - Cancellation
  - Timeouts
- Generates cross-platform client and server bindings for many languages
- Most common usage scenarios include connecting services in microservices style architecture and connecting mobile/browser clients to backend services

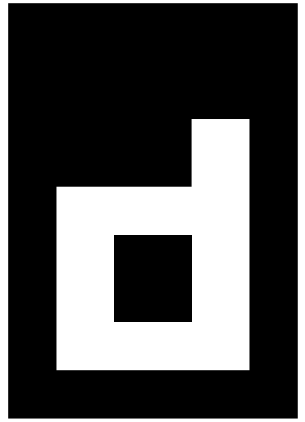
```
// specification of a horizontal pod autoscaler.
message HorizontalPodAutoscalerSpec {
  // reference to scaled resource; horizontal pod autoscaler will learn the current resource consumption
  // and will set the desired number of pods by using its Scale subresource.
  optional CrossVersionObjectReference scaleTargetRef = 1;

  // lower limit for the number of pods that can be set by the autoscaler, default 1.
  // +optional
  optional int32 minReplicas = 2;

  // upper limit for the number of pods that can be set by the autoscaler; cannot be smaller than MinReplicas.
  optional int32 maxReplicas = 3;

  // target average CPU utilization (represented as a percentage of requested CPU) over all the pods;
  // if not specified the default autoscaling policy will be used.
  // +optional
  optional int32 targetCPUUtilizationPercentage = 4;
}
```

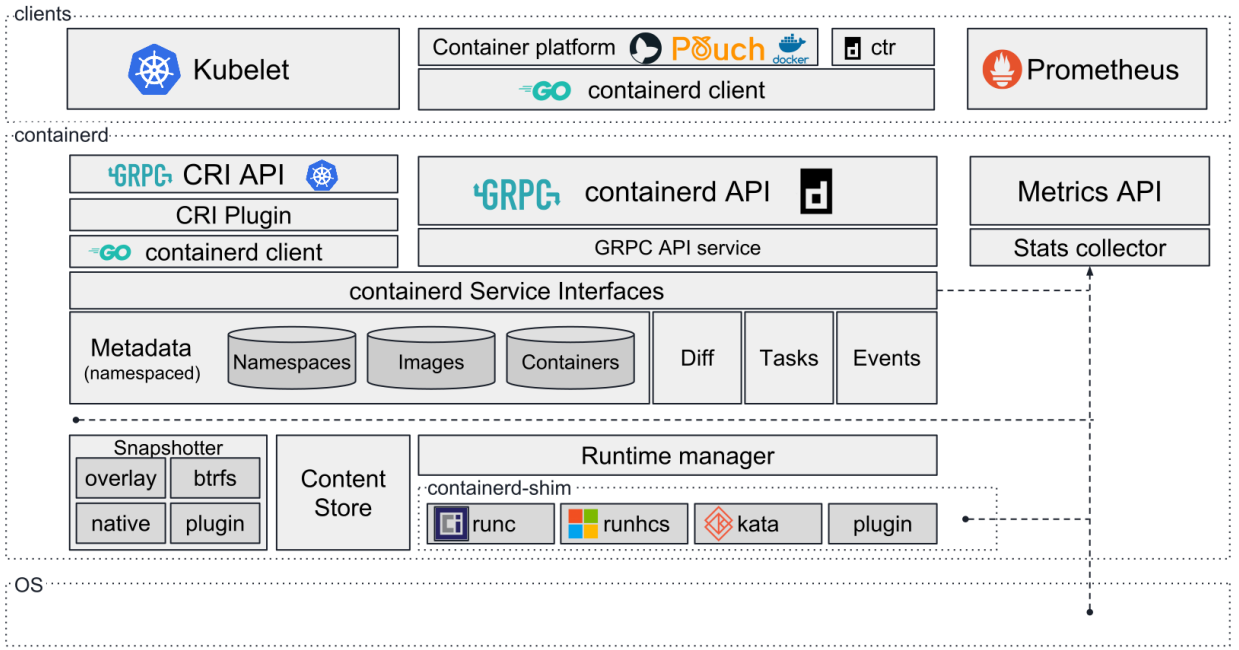
Step 1: Hack a gRPC Client and Server

- **What are we doing?**
  - Learning how to use an RPC system
  - Creating an interface in IDL
  - Building a Go microservice to impl it
  - Building a JavaScript client to test it
- **Jump into the Lab doc and complete step 1 !**

Docker and Containerd

- An industry-standard container runtime with an emphasis on:
  - Simplicity
  - Robustness
  - Portability
- The OCI container manager under Docker
- Has a gRPC (!) API
- Available as a daemon for Linux and Windows
- Manages the complete container lifecycle:
  - Image transfer
  - Image storage
  - Container execution
  - Container supervision
  - Low-level storage
  - Network attachments
  - and more!



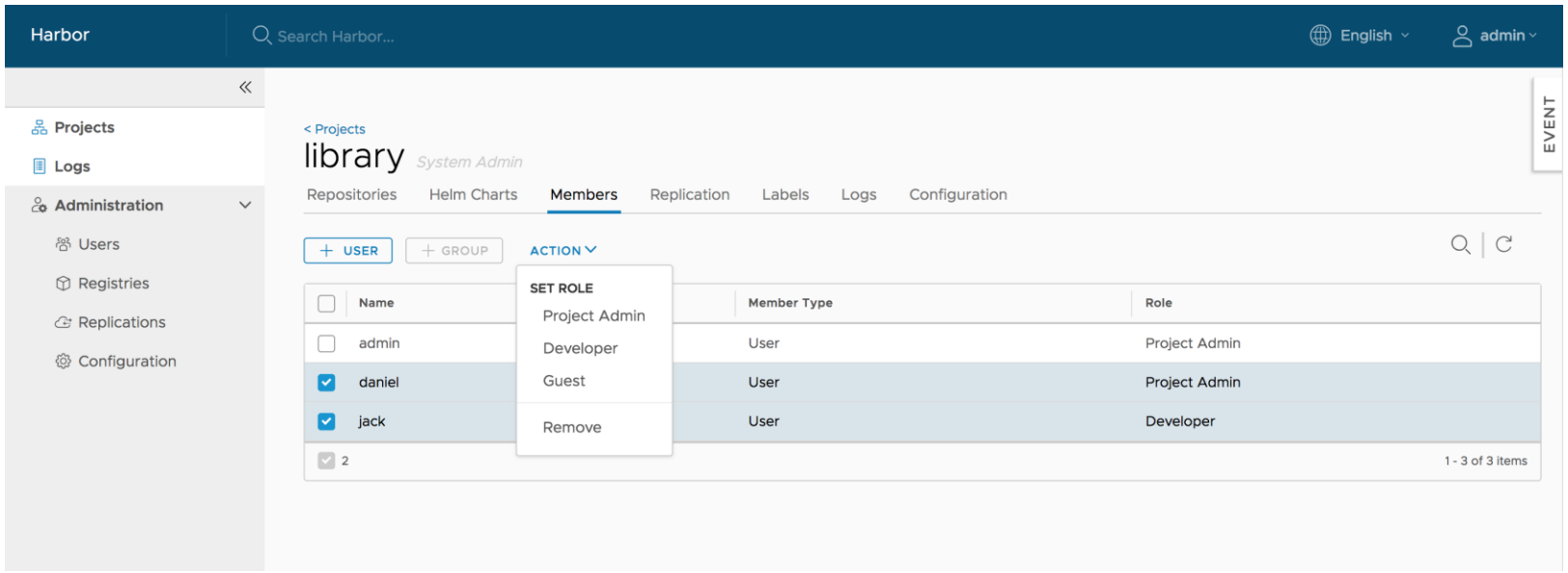


Step 2: Containerize that Microservice!

- **What are we doing?**
  - Containerizing our microservice
  - Installing Containerd
  - Running and managing containers with Containerd
    - … and ctr!
- **Jump into the Lab doc and complete step 2 !!**

Harbor

- Harbor is an open source container registry project
  - Stores containers
  - Signs containers
  - Scans container content
- Extends the open source Docker Distribution by adding the functionality usually required by users such as:
  - Security
  - Identity
  - Image management



Step 3: Push it

- **What are we doing?**
  - Pushing our containerized image to the Harbor registry service
  - Exploring the Harbor GUI
  - Pulling and running the image
- **Jump into the Lab doc and complete step 3 !!!**

Kubernetes

- Kubernetes is an open-source container-orchestration system for automating
  - Application deployment
  - Scaling
  - Management
- Originally designed by Google, and now maintained by the Cloud Native Computing Foundation

```yaml
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

Step 4: Roll it out

- **What are we doing?**
  - Using Kubernetes to deploy and manage our containerized microservice
  - Pulling the image from Harbor
  - Scaling it
- **Jump into the Lab doc and complete step 4 !!!!**

Helm

- Helm is the first application package manager designed for Kubernetes
- It allows users to describe application structure through convenient yaml based “helm-charts”
- Deployed applications can be managed with simple helm commands
- New application can be easily composed of existing loosely-coupled microservices
- Users deploying Helm charts can tailor them to their needs by settings variables in a values file
- The Helm chart template is combined with variable values to produce K8s specific configuration files

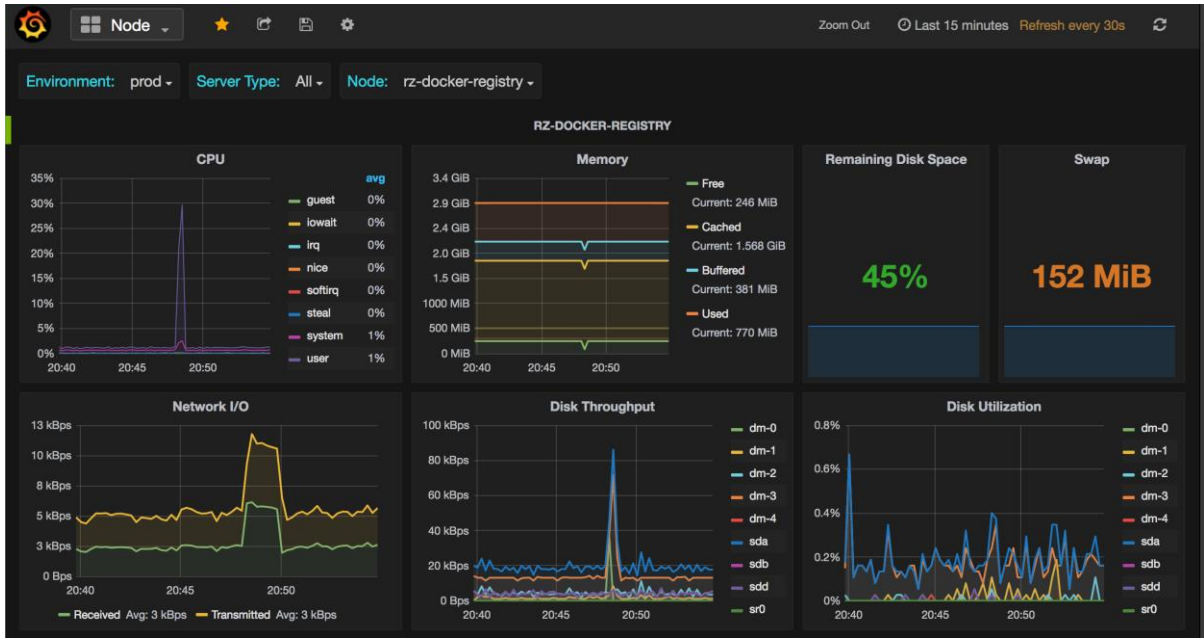

```
{{- if .Values.configmap }}
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "hmcts.releaseName" . }}
  labels:
    {{- ( include "labels" . ) | indent 4 }}
data:
  {{- range $key, $val := .Values.configmap }}
  {{ $key }}: {{ $val | quote }}
  {{- end}}
{{- end}}
```

Step 5: Package it

- **What are we doing?**
  - Using Helm to create a packaged solution for our microservice application
  - Deploying the application on a cloud hosted production cluster
- **Jump into the Lab doc and complete step 5 !!!!!**

Prometheus

- Prometheus is an open-source software project written in Go used to record real-time metrics in a time series database
- Uses an HTTP pull model (scrapping metrics from end points using the OpenMetrics format)
- Provides a flexible timeseries DSL for queries
- Supports real-time alerting
- Integrates deeply with K8s
- Easy to integrate with properly designed microservices
- Uses the Grafana web GUI as a front end



Step 6: Monitor it

- **What are we doing?**
  - Add some open metrics to your microservices
  - Scrape ‘em with Prom
- **Jump into the Lab doc and complete step 6 !!!!!!**

EFK

- Fluentd is a cross platform open-source data collection tool
  - Frequently used for
    - Log forwarding
    - Log aggregation
- Can be used in various roles to create a Unified Logging Layer (ULL)
- An open source software project originally developed at Treasure Data (now a part of ARM)
- Written primarily in Ruby programming language with core data processing elements in C for performance

```
<source>
  # Wordpress Database
  @type forward
  port 24000
  @label wordpress
</source>
<source>
  # Wordpress
  @type forward
  port 24100
  @label wordpress
</source>
<source>
  # Guestbook Database
  @type forward
  port 24200
  @label guestbook
</source>
<source>
  # Guestbook
  @type forward
  port 24300
  @label guestbook
</source>
<match **>
  @type stdout
</match>
<label wordpress>
  <match **>
    @type file
    path /tmp/wordpress-log
      <buffer>
        timekey 60s
        timekey_wait 1m
      </buffer>
  </match>
</label>
<label guestbook>
  <match **>
    @type file
    path /tmp/guestbook-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
  </match>
</label>
```

Step 7: Track it

- **What are we doing?**
  - Add some logging to your microservices
  - Forwarding the log events with Fluentd
- **Jump into the Lab doc and complete step 7 !!!!!!!**

Istio

- Istio is an open source service mesh that provides the key cross cutting concerns needed to successfully run a distributed microservice architecture
  - Mutual authentication
  - Service to service authorization
  - Traffic management
  - Tracing
  - Monitoring
  - Logging
  - Policy
  - Cluster Ingress
- Istio reduces the complexity of managing microservice deployments by providing a uniform way to deploy and manage these services

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

Step 8: Trace it

- **What are we doing?**
  - Using Istio to view context based activity in our application
- **Jump into the Lab doc and complete step 8 !!!!!!!!**

Telepresence

- Telepresence lets you run a single service locally while making it act as a component of a remote Kubernetes cluster
- This lets developers working on multi-service applications:
  - Do fast local development of a single service, even if that service depends on other services in the cluster
  - Make a change to the service, build, and immediately see the new service in action
  - Use any tool installed locally to test/debug/edit the service
    - Debuggers
    - IDEs
    - Etc.
- Telepresence works on:
  - Mac OS X
  - Linux


```
devlaptop$ # Let's say I have an nginx server running in Kubernetes:
devlaptop$ kubectl run --expose --port 80 mynginx --image=nginx
service "mynginx" created
deployment "mynginx" created
devlaptop$ # I'll start a Telepresence proxy in the Kubernetes cluster:
devlaptop$ kubectl run --port 8080 myserver --image=datawire/telepresence-k8s:0.41
deployment "myserver" created
devlaptop$ # I'll expose it to the Internet:
devlaptop$ kubectl expose deployment myserver --type=LoadBalancer --name=myserver
service "myserver" exposed
devlaptop$ # Next, I'll start a shell session whose contents will be proxied to Kubernetes:
devlaptop$ telepresence --deployment myserver --expose 8080 --run-shell
Starting proxy...
@gke_cluster|devlaptop$ # I will start a local web server on port 8080:
@gke_cluster|devlaptop$ echo "hello from my laptop" > demo.txt
@gke_cluster|devlaptop$ python3 -m http.server 8080 > /dev/null &
[1] 22323
@gke_cluster|devlaptop$ # Now let's find out the external IP for our service:
@gke_cluster|devlaptop$ kubectl get service myserver
NAME       CLUSTER-IP     EXTERNAL-IP       PORT(S)          AGE
myserver   10.3.242.226   104.197.103.123   8080:30022/TCP   50s
@gke_cluster|devlaptop$
```

Step 9: Debug it

- **What are we doing?**
  - Using Telepresence to debug runtime errors in a cloud native application
- **Jump into the Lab doc and complete step 9 !!!!!!!!!**

Thanks for attending!

@RandyAbernethy

randy@rx-m.com