CloudOps

To Russia, with Love

Kubernetes in Exotic Locations

Presented by:
Michael Wojcikiewicz
Container Solutions Architect, CloudOps

To Russia, with Love

How one rancher has found success in the Russian beef business

* Google search for “To Russia With Love Rancher…”

Agenda

1) Intro - myself, CloudOps and CNCF
2) The story so far...
3) International presence required
4) Problems
5) Solutions
6) Q & A

Intro: Michael Wojcikiewicz

- Container Solutions Architect at CloudOps
- Going on 20 years experience in IT
- Started as LAMP(erl) Stack developer, stints with Python, Java, NodeJS
- Worked at large enterprises and small startups throughout career
- Moved into DevOps role in last 5 years
- 3 years production experience with GKE
- ~1 year production experience with RKE
- Google Cloud Architect Certified Professional

CloudOps

Google

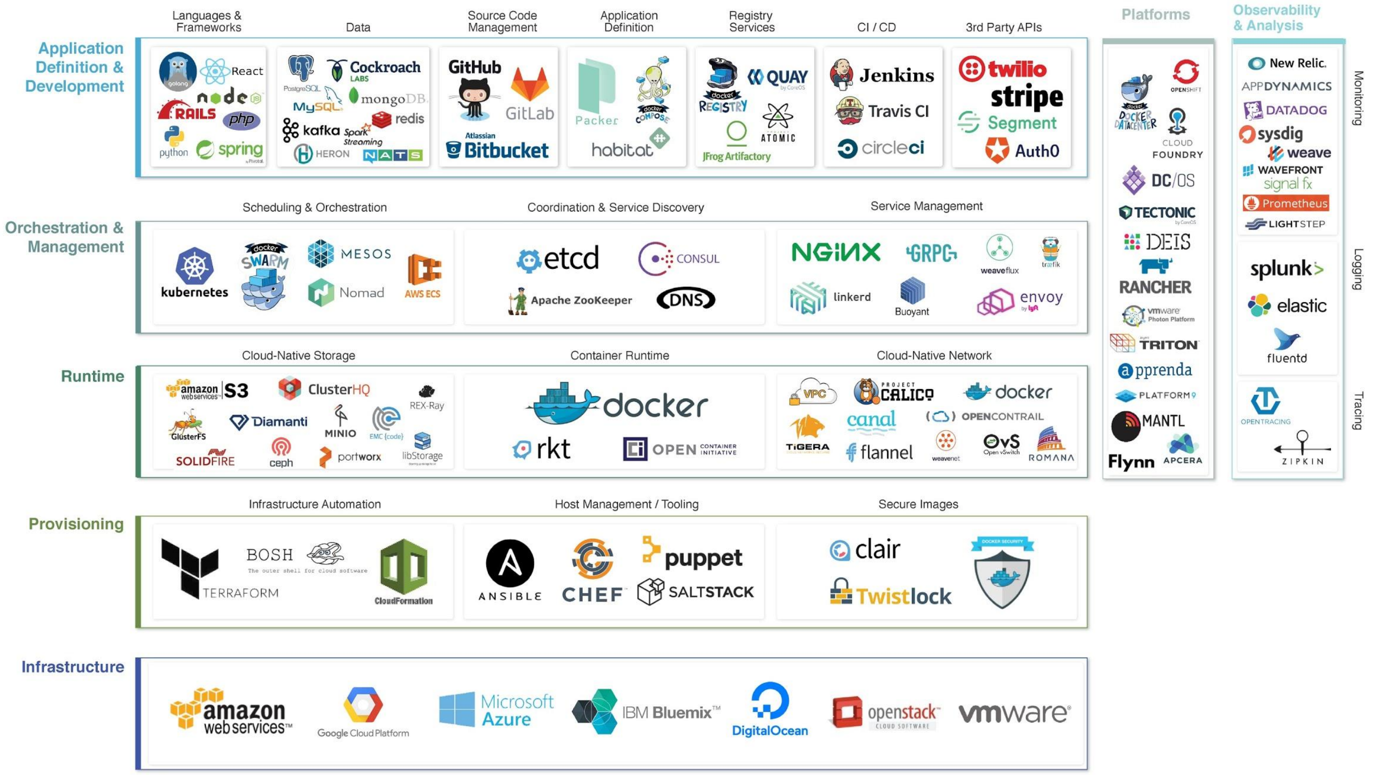
Certified Professional

Cloud Architect

Navigating the Landscape

Cloud Native Landscape v0.9.2

Redpoint | CLOUD NATIVE COMPUTING FOUNDATION



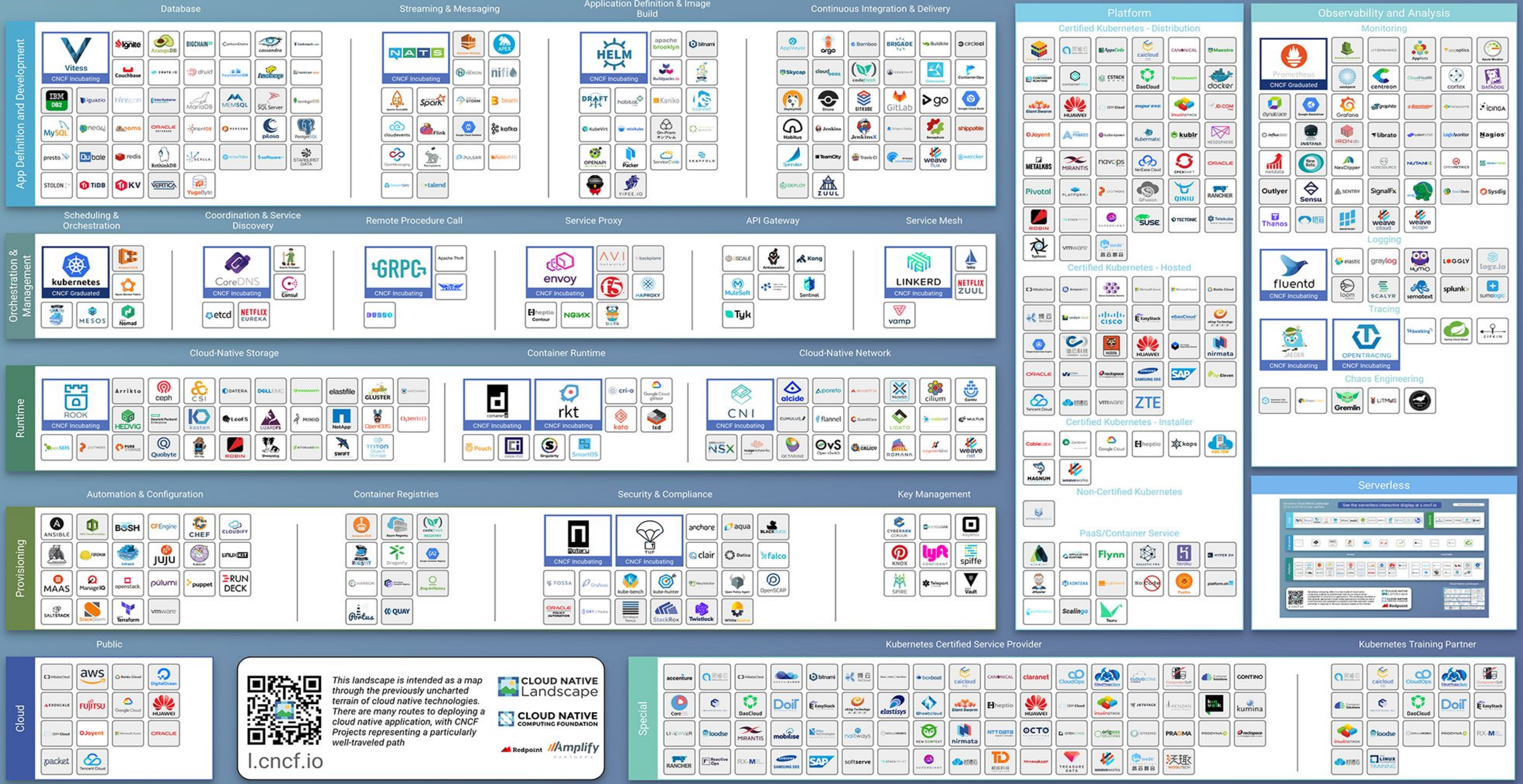
http://github.com/cncf/landscape @dankohn1 @lennypruss

Cloud Native Landscape

2018-10-23T00:25:01Z ca9f036

See the interactive landscape at l.cncf.io

Greyed logos are not open source



App Definition and Development

Database

Streaming & Messaging

Application Definition & Image Build

Continuous Integration & Delivery

Orchestration & Management

Scheduling & Orchestration

Coordination & Service Discovery

Remote Procedure Call

Service Proxy

API Gateway

Service Mesh

Runtime

Cloud-Native Storage

Container Runtime

Cloud-Native Network

Provisioning

Automation & Configuration

Container Registries

Security & Compliance

Key Management

Platform

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

Non-Certified Kubernetes

PaaS/Container Service

Observability and Analysis

Monitoring

Logging

Tracing

Chaos Engineering

Serverless

Cloud

Public

Special

Kubernetes Certified Service Provider

Kubernetes Training Partner

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path

l.cncf.io

CLOUD NATIVE Landscape

CLOUD NATIVE COMPUTING FOUNDATION

Redpoint

Amplify PARTNERS

Navigate the Landscape with Confidence!

DevOps Platform and Practices Assessment

An objective evaluation of:

- technical performance and security risks
- business impact of downtime, data loss, regionality, speed of innovation
- compliance requirements
- path (including gaps) to DevOps best practices and cloud native development and delivery

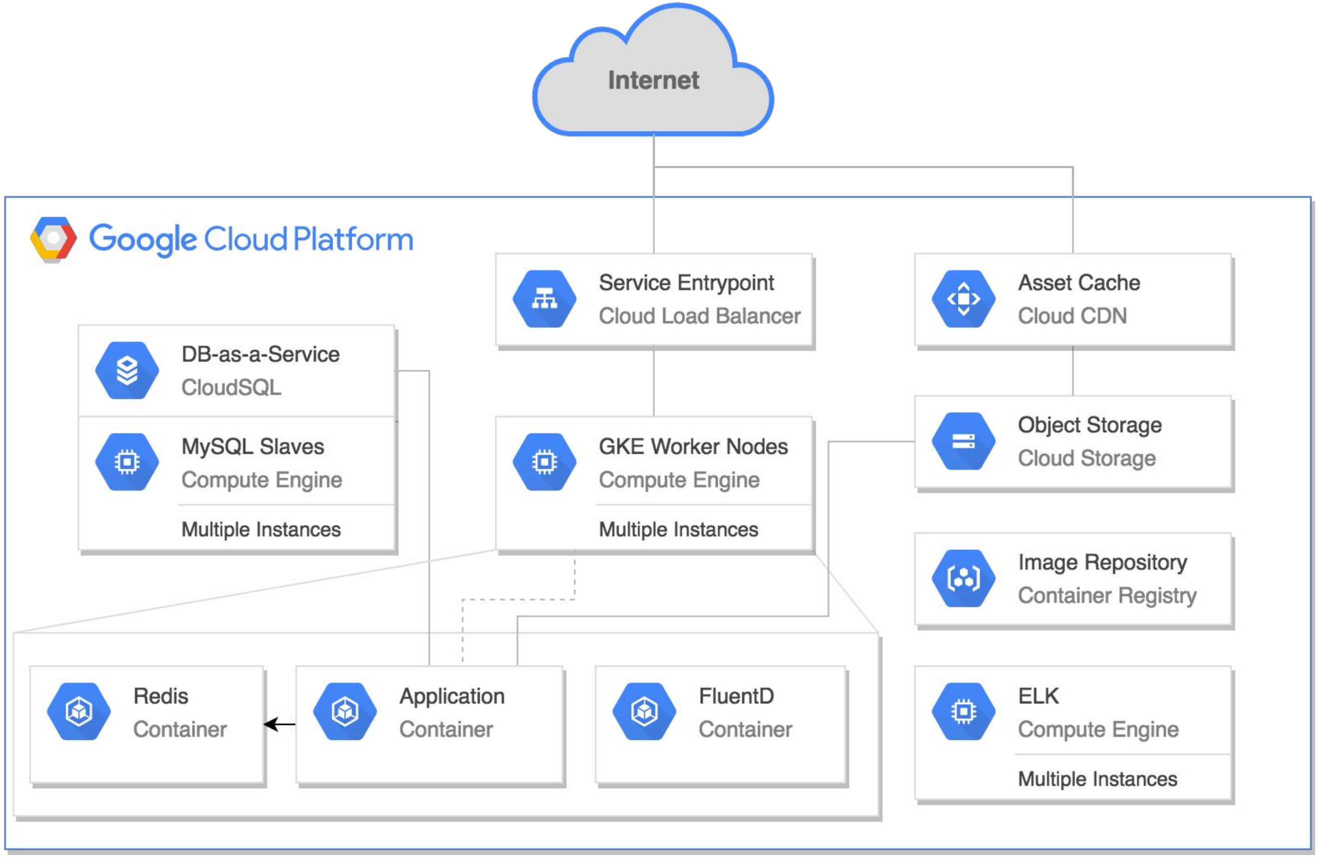
https://www.cloudops.com/devops-platform-and-practices-assessment/

info@cloudops.com

The Story So Far...

- Large European client in Hospitality space
- Many interconnected but independently developed and managed apps
- Migrated from AWS traditional 3 tier setup on VMs to containers in Google Kubernetes Engine (GKE) on Google Cloud Platform (GCP)
- Leveraged several other GCP-specific resources:
  - Object storage
  - Cloud CDN
  - Load balancers
  - Google Container Registry
  - CloudSQL

GCP Architecture - Single Component



International Presence Required

- Migration to GCP completed successfully (~March 2018)
- Since client is European, many customers are from the Russian Federation
- Russian Law 152-FZ requires Data Sovereignty in Russia for Russian customers

---

❖ As of this writing, there are no hyper cloud providers in Russia (no GCP, AWS or Azure)

❖ So now what?

Need an “on prem” solution. Let’s find a “prem” first!

Cloud Providers in Russia

- Team investigated several cloud providers in Russia
  - Communications proved difficult (English/Russian, Gmail translate surprisingly useful though!)
  - Some providers very sketchy and non-starters

- Settled on a VMWare CloudDirector-based provider
  - Full 152-FZ compliant env!
  - Terraform plugins from VMWare did not work
  - Third party TF provider worked “sometimes”
  - Issues likely relating to connectivity between development location (NA, EU) and Russian provider, and outdated version of the VMWare products
  - Certain features advertised in the provider did not work, notably specifying the CPU resources, which made it fairly useless

In Soviet Russia...

- Ended up creating 2 flavours of Debian 9 base images with different root sizes to fill all roles of VMs in Russian deployment
  - Very manual process, but at least no longer blocked on getting resources local to Russia
- The best part?
  - Old version of Cloud Director
  - FLASH BASED ADMINISTRATIVE UI
  - Recommended to use Internet Explorer so all UI features work
  - !!!!
  - !!!
  - !!
  - !

Needless to say: this was run in isolated VM with nothing else on it

Which Kubernetes Distribution/ Installation Tool?

- KubeSpray, kubeadm considered, dismissed
  - Complex to operate after installation (ie. upgrades)
  - Complex configurations
  - Lengthy installation and configuration times
  - Historically had issues with multi-master setups

Enter: Rancher Kubernetes Engine

- Rancher Kubernetes Engine (RKE) to the rescue
  - IMNSHO the best custom Kubernetes installer to date
    - Great analysis here: https://medium.com/@cfatechblog/bare-metal-k8s-clustering-at-chick-fil-a-scale-7b0607bd3541
  - RKE requirements:
    - Docker (compatible w/Kubernetes version)
    - SSH (and a passwordless keyed login)
    - Nothing else

Rancher Kubernetes Engine

- A single configuration YAML file to specify Kubernetes cluster configuration
- A single go binary ‘rke’ to manage RKE
- ‘rke up’ sets up SSH tunnels to designated hosts
    - Uses pre-made Docker images from Rancher to run all the Kubernetes components (ie. API-server, kubelet, etc.)
    - Bootstraps a new Kubernetes cluster with the desired config in under 5 minutes
    - Multi-master works out of the box!
    - Once provisioned, KUBECONFIG client certificate is output by the ‘rke’ binary, to be used by kubectl for interacting with the cluster

Minimal RKE cluster.yml Configuration

```
nodes:
- address: masterworker
  port: "22"
  role:
  - controlplane                               # this node is a member of the control plane
  - etcd                                       # this node is a member of the etcd datastore backing k8s
  - worker                                     # this node is a worker node
services:
  kube-api:
    service_cluster_ip_range: 10.43.0.0/21     # service CIDR
  kube-controller:
    cluster_cidr: 10.42.0.0/21                 # pod CIDR
    service_cluster_ip_range: 10.43.0.0/21     # service CIDR
  kubelet:
    cluster_dns_server: 10.43.0.10             # DNS service location, service CIDR + 11 addresses
network:
  plugin: canal                                # Calico + Flannel CNI
authentication:
  strategy: x509                               # Client Cert auth only
ssh_key_path: "/path/to/key"
authorization:
  mode: rbac                                   # enable RBAC
ignore_docker_version: true                    # for non certified, but nonetheless working versions of
Docker
cluster_name: "my-cluster"
```

Operating a Rancher Kubernetes Engine Cluster

- Lifecycle management is simple:
    - Adding new nodes? Just add to the cluster.yaml and run ‘rke up’ again
    - Decommissioning nodes? Just remove them from cluster.yaml and run ‘rke up again’
    - Upgrading Kubernetes components? Change the Docker Image version you desire
        - Ie. fix a specific component of Kubernetes does not require upgrading entire cluster
    - Upgrading Kubernetes entirely? Grab new ‘rke’ binary, adjust cluster.yaml if you have pinned version, run ‘rke up’
        - Rolling upgrade of each node, just like cloud providers do! (Taint, Drain, Upgrade, Untaint)
        - **Best Practices Tip:** don’t upgrade over a major version (ie. from 1.9 - 1.11), always do the intermediate upgrades (1.9 -> 1.10 -> 1.11)

Hang on a Second, It’s Not THAT Simple…

- RKE is great for provisioning and operating on-prem cluster
- Not a complete solution in itself
- Notably missing:
  - Storage provider (CSI) to enable persistent volumes and persistent volume claims
  - Load balancer
- Other missing features from GCP:
  - Google Container Registry
  - Object storage
  - CloudCDN
  - CloudSQL

Google Container Registry On-Prem

- Since images do not contain customer data, Docker Registry does not need to be duplicated in Russia
- With a proper Kubernetes secret, it’s easy to enable remote pulling of images from GCR outside of GCP:

```
apiVersion: v1
data:
  .dockerconfigjson: BASE64_ENCODED_SERVICE_ACCOUNT_JSON_KEY
kind: Secret
metadata:
  name: my-gcr-secret
  namespace: default
type: kubernetes.io/dockerconfigjson
```

- And then use that secret in your deployment manifest

```
spec:
  imagePullSecrets:
  - name: my-gcr-secret
```

CloudSQL On-Prem

- Nothing special here, just install MySQL
- Same Debian 9 image used in Kubernetes cluster
- Used existing Ansible playbooks to configure MySQL 5.7
    - 1 Master
    - 2 Slaves
        - Realtime slave (for reporting and failover)
        - Delayed slave - ship binlogs, but does not apply for 30 days
            - Allows to restore to point in time even with replicated corruption

Object Storage / CDN on prem

DISCLAIMER: Rook/Ceph was not production ready when these decisions were made

- Decided to use GlusterFS to provide object storage-like functionality
- 2 NGINX Ingress nodes to serve content from GlusterFS to act at CDN, as well as reverse proxy the Kubernetes cluster service NodePorts and Ingress ports
  - Each ingress node gets replicated data in GlusterFS on a dedicated device brick (100GB)
  - Serves data from GlusterFS on a specific path = insta-poormans-CDN
- GlusterFS volume is mounted on Kubernetes worker nodes for read/write
- GlusterFS volume is added to deployment as HostPath volume under /cdn-data so each app pod can write content for CDN

Adequate solution for object storage and a CDN

Redis... needs PVC

- Redis is being used by app as session cache
- Since we have no PVC ability, we temporarily re-configured the application to use the SQL database as session cache

BUT... Now that Rook graduated to Incubating in CNCF (Dec 2018), let’s use that!

- Ceph is most mature storage driver for Rook
- Installed using ‘helm’ (rook-stable/rook-ceph)

```
helm repo add rook-stable https://charts.rook.io/stable

helm install --namespace rook-ceph-system rook-stable/rook-ceph
```

Rook and RKE Issues

- Since Rook is a CSI (Container Storage Interface) driver, it’s loaded dynamically by Kubernetes, and expects the Kubernetes volume plugin dir (aka Kubelet Plugins) to be writable and registered into the Kubernetes engine
- It’s a well known problem, with an easy fix, simply add this to the cluster.yaml’s ‘kubelet’ section

```
kubelet:
 extra_args:
  volume-plugin-dir: /usr/libexec/kubernetes/kubelet-plugins/volume/exec
 extra_binds:
  -
/usr/libexec/kubernetes/kubelet-plugins/volume/exec:/usr/libexec/kubernetes/kubelet-plugins/volume/exec
```

- Then run ‘rke up’ which patched the live cluster, and when it was done, all Rook related pods had restarted and it worked as expected

Creating a Ceph Cluster in Rook

- Rook Operator is installed, now to configure the Ceph Cluster
- Each worker node has dedicated ‘sdb’ 100GB SSD brick
- Vanilla ‘cluster.yaml’ from the rook source tree examples, with following modification to only use the brick for PVCs

```
storage: # cluster level storage configuration and selection
    useAllNodes: true
    useAllDevices: false
    deviceFilter: sdb
```

- Run kubectl create -f cluster.yaml

Kubernetes StorageClass

Rook operator now operates the Ceph cluster with the designated storage bricks

- Now we need a Kubernetes StorageClass from Rook/Ceph to actually be able to create PVCs
- ‘storageclass.yaml’ from the Rook source tree examples will create a Ceph ReplicaPool and a Kubernetes storage class that uses it

Ceph ReplicaPool and Kubernetes StorageClass

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook-ceph
spec:
  replicated:
    size: 1
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: rook-ceph-block
provisioner: ceph.rook.io/block
parameters:
  blockPool: replicapool
  clusterNamespace: rook-ceph
  fstype: xfs
```

Redis... with PVC!

- Now we have a StorageClass called ‘rook-ceph-block’
- Let’s use Helm version of Redis, and tell it to use this StorageClass
- Redis has 2 components that require PVC: ‘master’ and ‘slave’, and lets you set command-line (or values file) overrides for those:

```
helm install --name=redis stable/redis \
  --set master.persistence.storageClass=rook-ceph-block \
  --set slave.persistence.storageClass=rook-ceph-block
```

What About Logging?

FluentD is the defacto standard logging agent for Kubernetes

- Much needed in the logging space
- Implemented in Ruby (non dynamic language based agent in development)
- Many input plugins, many output plugins
- Has been under active development, with some breaking (frustrating!) changes when it went from 0.x to 1.x, and weird breakage in 1.2.x packages
- Our requirements:
    - Comprehensive Kubernetes monitoring configuration
    - Sane application log parsing out of the box (ie. JSON in JSON log line)
    - Sink to ElasticSearch (E from ELK)

FluentD

Best FluentD image:

```
fluent/fluentd-kubernetes-daemonset:v1.1.3-debian-elasticsearch
```

- 1.2.x has a dependency on a non standard Debian library existing on the worker node, causes super high load if it's not there
- The out of the box JSON in JSON parsing didn't work (and my pleading with the FluentD engineers at KubeCon to fix it in the image appears to be unanswered) and it was suggested to patch in a new config to address this
- Also needed to designate certain attributes in the JSON as numeric, so ElasticSearch indexes them the same way for searching

FluentD Configuration

Started with vanilla ‘kubernetes.conf’ from the FluentD Image, and added

```
<filter kubernetes.var.log.containers.**>
      @type parser
      <parse>
        @type json
        json_parser json
        types elapsed_time:float,status_code:integer,bytes_sent:integer
      </parse>
      replace_invalid_sequence true
      emit_invalid_record_to_error false
      key_name log
      reserve_data true
    </filter>
```

To do JSON in JSON parsing. The entire file was then added as a ConfigMap called ‘fluentd-config’ with a ‘kubernetes.conf’ being the single key, containing the contents of the config file

Run FluentD with Custom Configuration

Now simply patch the FluentD DaemonSet spec:

```
volumes:
 - name: fluentd-config
  configMap:
   name: fluentd-config
```

And map it into the FluentD pod (without squashing other conf files):

```
volumeMounts:
 - name: fluentd-config
  subPath: kubernetes.conf
  mountPath: /fluentd/etc/kubernetes.conf
```

Run FluentD with Custom Configuration

And finally, define the ‘FLUENT_ELASTICSEARCH_LOGSTASH_PREFIX’ environment variable to specify which ElasticSearch index the logs should go into

```
env:
  - name: FLUENT_ELASTICSEARCH_LOGSTASH_PREFIX
    value: "myapp"
```

- Now all logs flowing from this agent will go into the ‘myapp-($date)’ which rolls every day to a new index name
- This is useful if you are logging from many clusters to shared logging infrastructure to keep logging data organized

ELK? There's a Chart for That

Initially, the Russian installation had a separate dedicated ELK cluster (3 VMs for HA).

- Very underutilized / overprovisioned
- Could we use the compute resources in the Kubernetes cluster instead, and run ELK inside it?
- YES! - Helm chart: stable/elastic-stack

ELKing the Cluster

So we wiped the ELK cluster

- A fresh Debian 9 image
- Added the old ELK nodes to the RKE ‘cluster.yaml’ file as ‘worker’ nodes
- Ran ‘rke up’
- A couple minutes later had added the compute capacity from the ELK cluster to the Kubernetes cluster

Now to install the ELK chart, using our ‘rook-ceph-block’ StorageClass

```
helm install --name elk stable/elastic-stack \
  --set elasticsearch.data.persistence.size=50Gi \
  --set elasticsearch.data.persistence.storageClass=rook-ceph-block \
  --set elasticsearch.master.persistence.storageClass=rook-ceph-block
```

ELK - Cleaning Up

There is a small bug in the Helm Chart relating to Kibana (the ELK UI) and where to find the elasticsearch client service. One of these days I'll send a PR upstream to fix it.   In the meantime here is the fix (deployment name may vary depending on name you used to install):

```
kubectl edit deployment elk-kibana
```

- Change the 'ELASTICSEARCH_URL' env var value to match the service name:

```
env:
  - name: ELASTICSEARCH_URL
    value: http://elk-elasticsearch-client:9200
```

- Now change the FluentD Deployment YAML and update the 'FLUENT_ELASTICSEARCH_HOST' to reflect the same service name

Accessing ELK

Accessing ELK can now be done in several ways:

- kubectl port-forward svc/elk-kibana 5601:443
- Exposing the Kibana service as a NodePort, add to NGINX Ingress nodes as another host
- Kubernetes Ingress Resource with user/pass on https (best choice)

Speaking of Ingress...

RKE installs ‘nginx-ingress-controller’ as the default Ingress controller

- Use annotations to use nginx-controller and to secure service, if necessary.
  - ‘htpasswd’ can be used to create an auth string secret NGINX can use for HTTP Auth

```
$ htpasswd -nb elk changeme
elk:$apr1$O/QOEMkP$GXbki1OsdsQYoSW1uImQg1
```

Now put that string into a secret

```
kubectl create secret generic myapp-http-auth-secret
--from-literal=auth=elk:$apr1$O/QOEMkP$GXbki1OsdsQYoSW1uImQg1
```

- And if you have an SSL cert handy:

```
kubectl create secret tls tls-secret --cert myapp.cer --key myapp.key
```

Full Secured Ingress Example

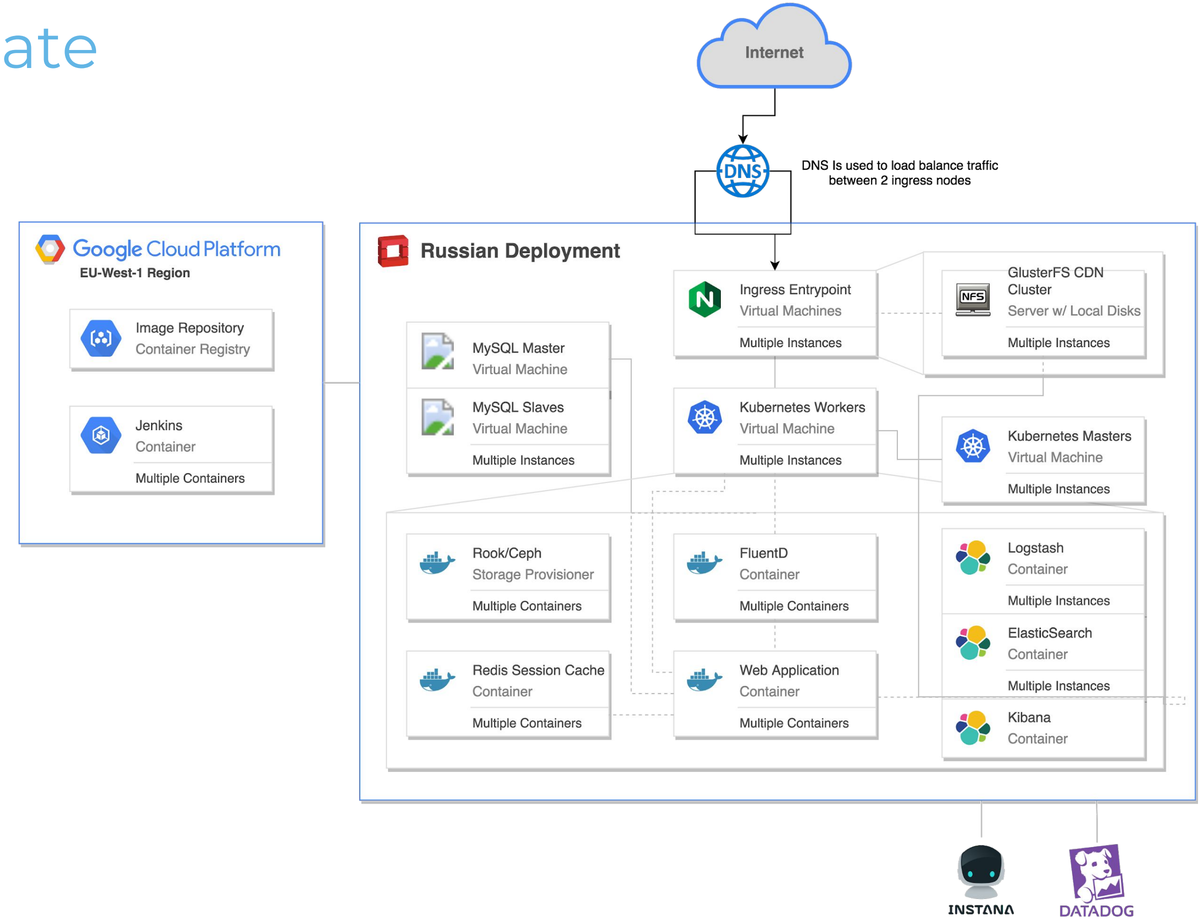
```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/auth-realm: Authentication Required - Kibana
    nginx.ingress.kubernetes.io/auth-secret: my-app-auth-secret
    nginx.ingress.kubernetes.io/auth-type: basic
  name: elk-k8s
  namespace: default
spec:
  rules:
  - host: elk.myapp.domain
    http:
      paths:
      - backend:
          serviceName: elk-kibana
          servicePort: 5601
        path: /
  tls:
  - hosts:
    - elk.myapp.domain
    secretName: tls-secret
```

Load Balancing… Ugh

For whatever reason, the Russian “cloud provider” does not have the ability to assign a hardware load balancer to front the infrastructure.

- In interest of time and effort the solution became
    - 2 NGINX Ingress nodes
    - Round robin DNS resolution for each nodes IP
    - Adequate solution as a “poor man’s load balancer”
    - Reverse Proxies all traffic on 80 and 443 (and serves CDN traffic on another path)
    - No SSL termination - handled by NGINX Ingress controller in Kubernetes instead

End State


Internet
DNS
DNS Is used to load balance traffic between 2 ingress nodes
Google Cloud Platform
EU-West-1 Region
Image Repository
Container Registry
Jenkins
Container
Multiple Containers
Russian Deployment
Ingress Entrypoint
Virtual Machines
Multiple Instances
GlusterFS CDN Cluster
NFS
Server w/ Local Disks
Multiple Instances
MySQL Master
Virtual Machine
MySQL Slaves
Virtual Machine
Multiple Instances
Kubernetes Workers
Virtual Machine
Multiple Instances
Kubernetes Masters
Virtual Machine
Multiple Instances
Rook/Ceph
Storage Provisioner
Multiple Containers
FluentD
Container
Multiple Containers
Logstash
Container
Multiple Instances
ElasticSearch
Container
Multiple Instances
Redis Session Cache
Container
Multiple Containers
Web Application
Container
Multiple Containers
Kibana
Container
INSTANA
DATADOG

Q & A

Special Thanks to

Julia Simon, Senior Product Marketing Manager at CloudOps

Emma De Angelis, Freelance Artist <design@emmadeangelis.com>

for making my chicken scratch of a presentation look professional!

More Info

- CloudOps - https://www.cloudops.com info@cloudops.com
- Rook - https://github.com/rook/rook
- RKE - https://github.com/rancher/rke
- FluentD - https://github.com/fluent/fluentd

Now….. Q&A?

Own your destiny in the cloud

Partner with CloudOps to transform, support and evolve your DevOps and cloud native practice

Cloud and code agnostic, but opinionated