Altinity

ClickHouse on Kubernetes!

Alexander Zaitsev, Altinity

Limassol, May 7th 2019

Altinity Background

- Premier provider of software and services for ClickHouse
- Incorporated in UK with distributed team in US/Canada/Europe
- US/Europe sponsor of ClickHouse community
- Offerings:
    - 24x7 support for ClickHouse deployments
    - Software (Kubernetes, cluster manager, tools & utilities)
    - POCs/Training

What is Kubernetes?

“Kubernetes is the new Linux”

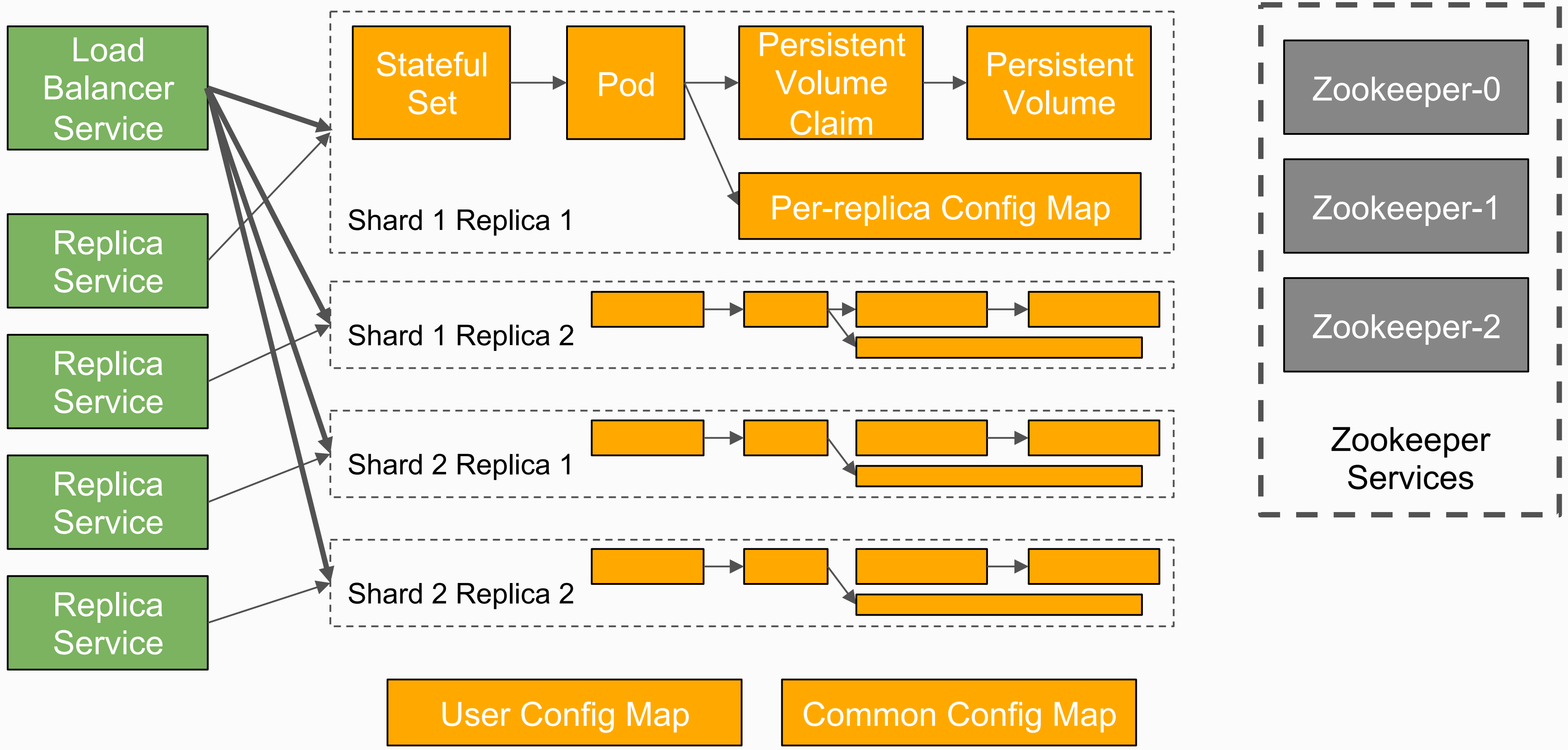
Actually it’s an open-source platform to:

- manage container-based systems
- build distributed applications declaratively
- allocate machine resources efficiently
- automate application deployment

Why run ClickHouse on Kubernetes?

1. Other applications are already there
2. Portability
3. Bring up data warehouses quickly
4. Easier to manage than deployment on hosts

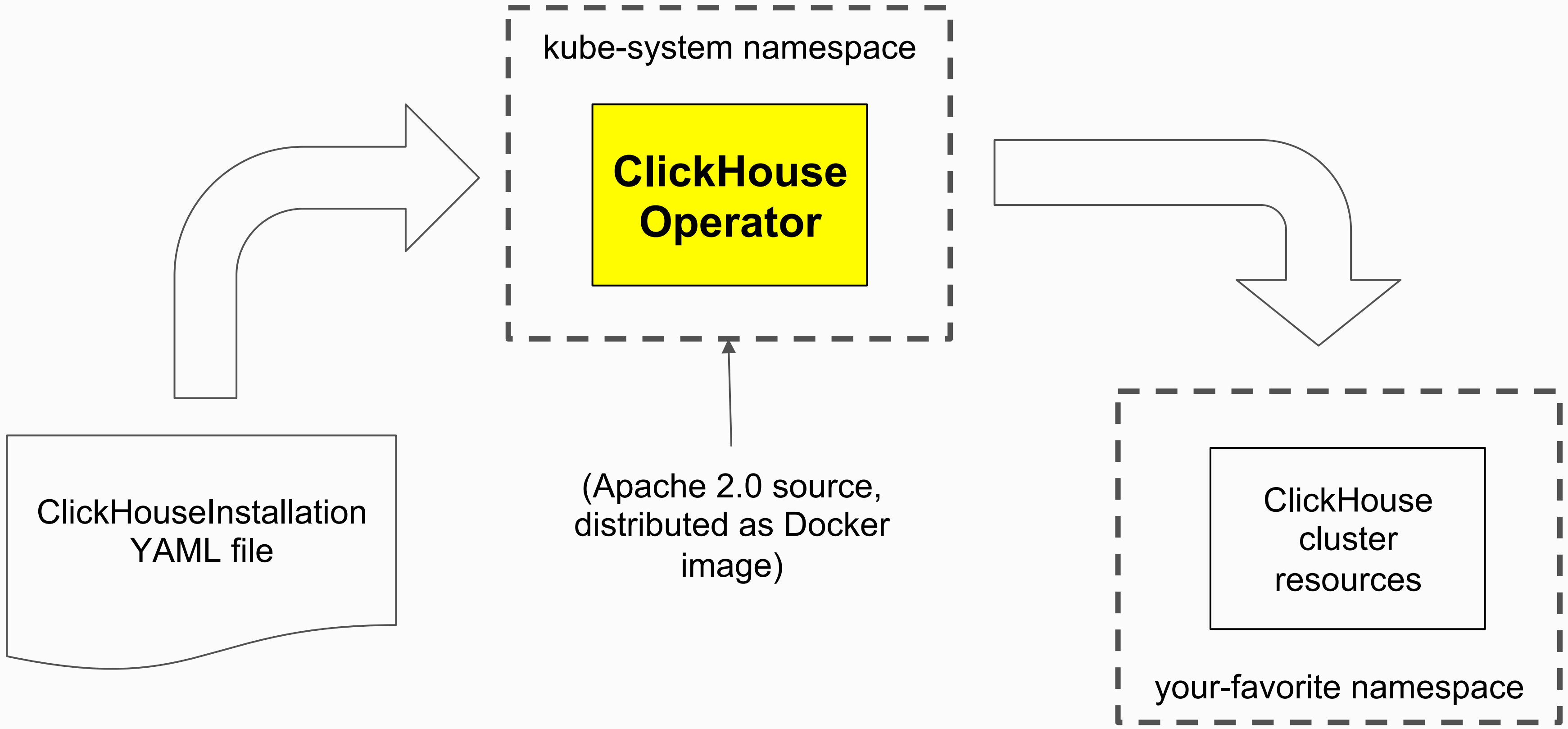
What does ClickHouse look like on Kubernetes?



Challenges running ClickHouse on Kubernetes?

1. Provisioning
2. Persistence
3. Networking
4. Transparency

The ClickHouse operator turns complex data warehouse configuration into a single easy-to-manage resource


kube-system namespace
ClickHouse Operator
ClickHouseInstallation YAML file
(Apache 2.0 source, distributed as Docker image)
ClickHouse cluster resources
your-favorite namespace

Altinity ClickHouse Operator Quick Start

Installing and removing the ClickHouse operator

[Optional] Get sample files from github repo:

```
git clone https://github.com/Altinity/clickhouse-operator
```

Install the operator:

```
kubectl apply -f clickhouse-operator-install.yaml
```

or:

```
kubectl apply -f https://raw.githubusercontent.com/Altinity/
clickhouse-operator/master/manifests/operator/clickhouse-
operator-install.yaml
```

Let’s start with a single-node cluster

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo-01"
spec:
  configuration:
    clusters:
       - name: "demo-01"
         layout:
           type: Standard
           shardsCount: 1
           replicasCount: 1
```

WARNING: This installation lacks persistent storage

See examples in later slides for storage definition

kubectl is our tool

```
$ kubectl create namespace demo
namespace/demo created

$ kubectl apply -f docs/examples/demo-01.yaml -n demo
clickhouseinstallation.clickhouse.altinity.com/demo-01 created

$ kubectl get all -n demo
NAME                         READY     STATUS     RESTARTS    AGE
pod/chi-a82946-2946-0-0-0    1/1       Running    0           52s

NAME                          TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)
AGE
service/chi-a82946-2946-0-0   ClusterIP      None             <none>         8123/TCP,9000/TCP,9009/TCP
52s
service/clickhouse-demo-01    LoadBalancer   10.108.198.248   <pending>      8123:31609/TCP,9000:32086/TCP
52s

NAME                                      DESIRED    CURRENT    AGE
statefulset.apps/chi-a82946-2946-0-0      1          1          52s
```

Next let’s add a shard

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo-01"
spec:
  configuration:
    clusters:
      - name: "demo-01"
        layout:
          type: Standard
          shardsCount: 2
          replicasCount: 1

$ kubectl apply -f docs/examples/demo-01.yaml -n demo
clickhouseinstallation.clickhouse.altinity.com/demo-01 configured
```

How to access your ClickHouse data warehouse on Kubernetes

Connect from within Kubernetes using service DNS name

```
# Use load balancer
clickhouse-client --host clickhouse-demo-01.test
# Connect to specific node
clickhouse-client --host chi-a82946-2946-0-0.test
# Connect via pod entry
kubectl exec -it chi-a82946-2946-0-0-0 -n demo -- clickhouse-client
```

Connect from outside Kubernetes using Ingress or Nodeport

```
# Kops deployment on AWS configures external ingress.
clickhouse-client --host $AWS_ELB_HOST_NAME
```

Replication requires Zookeeper to be enabled

Install minimal Zookeeper in separate namespace.

```
kubectl create ns zoons
kubectl apply -f zookeeper-1-node.yaml -n zoons
watch kubectl -n zoons get all
```

Note ZK node DNS name: **zookeeper-0.zookeepers.zoons**

You can also install using helm *or* use external ZK cluster

After inserting a ‘zookeepers’ clause we can add replicas

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo-01"
spec:
  configuration:
    zookeeper:
      nodes:
        - host: zookeeper-0.zookeepers.zoons
          port: 2181
    clusters:
      - name: "demo-01"
        layout:
          type: Standard
          shardsCount: 2
          replicasCount: 2
```

TIP: Confirm the DNS name of Zookeeper from with a pod

NOTE: Non-replicated tables do not replicate automatically when replicas are added

We can add and modify users with the ‘users’ clause

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo-01"
spec:
  configuration:
    users:
      demo/default: secret
      demo/password: demo
      demo/profile: default
      demo/quota: default
      demo/networks/ip: "::/0"
    clusters:
        - name: "demo-01"
          layout:
            type: Standard
            shardsCount: 2
            replicasCount: 1
```

TIP: User and profile changes take a few minutes to propagate. Confirm changes using clickhouse-client

To make storage persistent and set properties add an explicit volume claim template with class and size

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "storage"
spec:
  defaults:
    deployment:
      volumeClaimTemplate: storage-vc-template
  templates:
    volumeClaimTemplates:
      - name: storage-vc-template
        persistentVolumeClaim:
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 2Gi
  configuration:
```

TIP: Check syntax carefully as errors may result in failures to allocate or mount volumes

TIP: Confirm storage by ‘kubectl exec’ into pod; run ‘df -h’ to confirm mount

storageClassName can be used to set the proper class of storage as well as disable dynamic provisioning

Use kubectl to find available storage classes:

```
kubectl describe StorageClass
```

Bind to default storage:

```
spec:
  storageClassName: default
```

Bind to gp2 type

```
spec:
  storageClassName: gp2
```

Disable dynamic provisioning and use static PVs:

```
spec:
  storageClassName: "
```

Set the ClickHouse version using a podTemplate

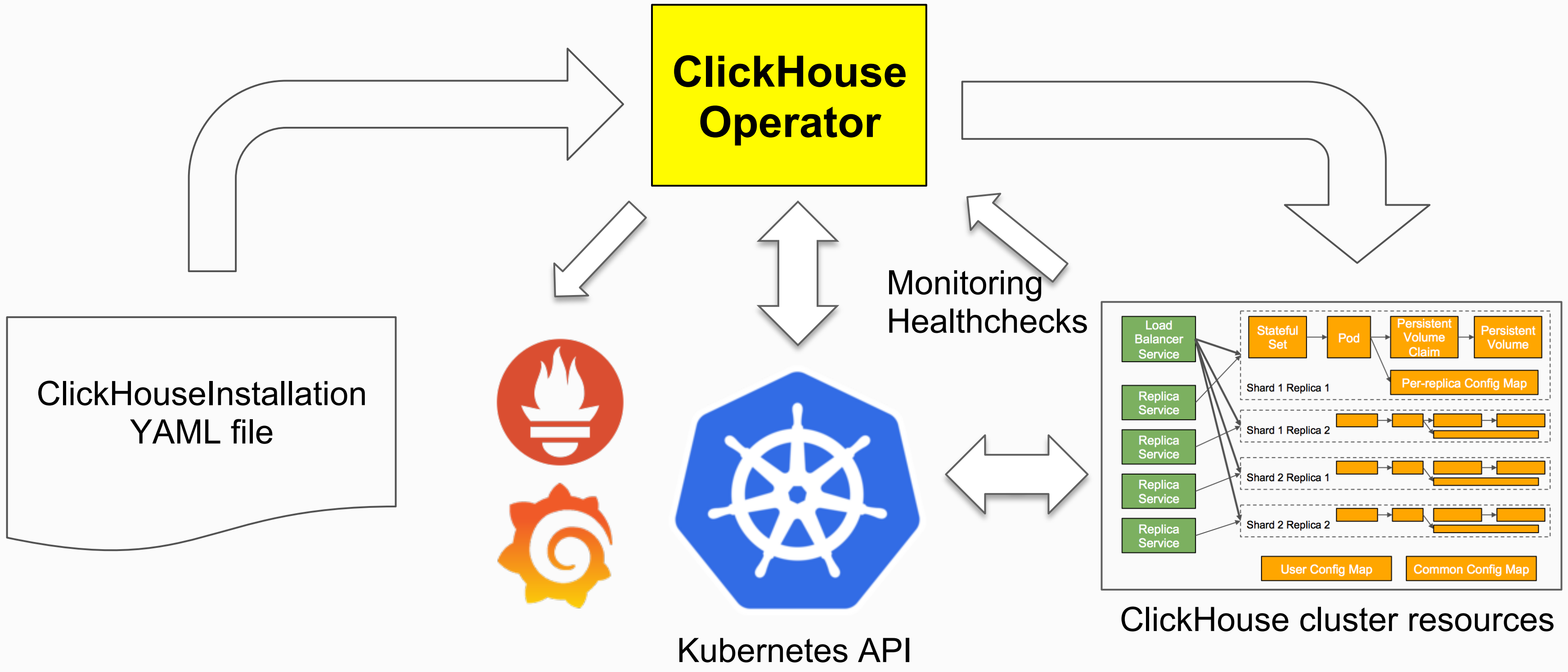
```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo-02"
spec:
  defaults:
    deployment:
      podTemplate: clickhouse-stable
      volumeClaimTemplate: storage-vc-template
  templates:
    podTemplates:
    - name: clickhouse-stable
      containers:
      - name: clickhouse
        image: yandex/clickhouse-server:19.4.3.11
    volumeClaimTemplates:
# Etc.
```

TIP: Always specify the image version fully; do not use ‘latest’ tag

More pod template tricks: controlling resources

```
spec:
  defaults:
    deployment:
      podTemplate: clickhouse-stable
      volumeClaimTemplate: storage-vc-template
  templates:
    podTemplates:
    - name: clickhouse-stable
      containers:
      - name: clickhouse
        image: yandex/clickhouse-server:19.4.3.11
          resources:
            requests:
              memory: "512Mi"
              cpu: "500m"
             limits:
              memory: "512Mi"
                cpu: "500m"
# Etc.
```

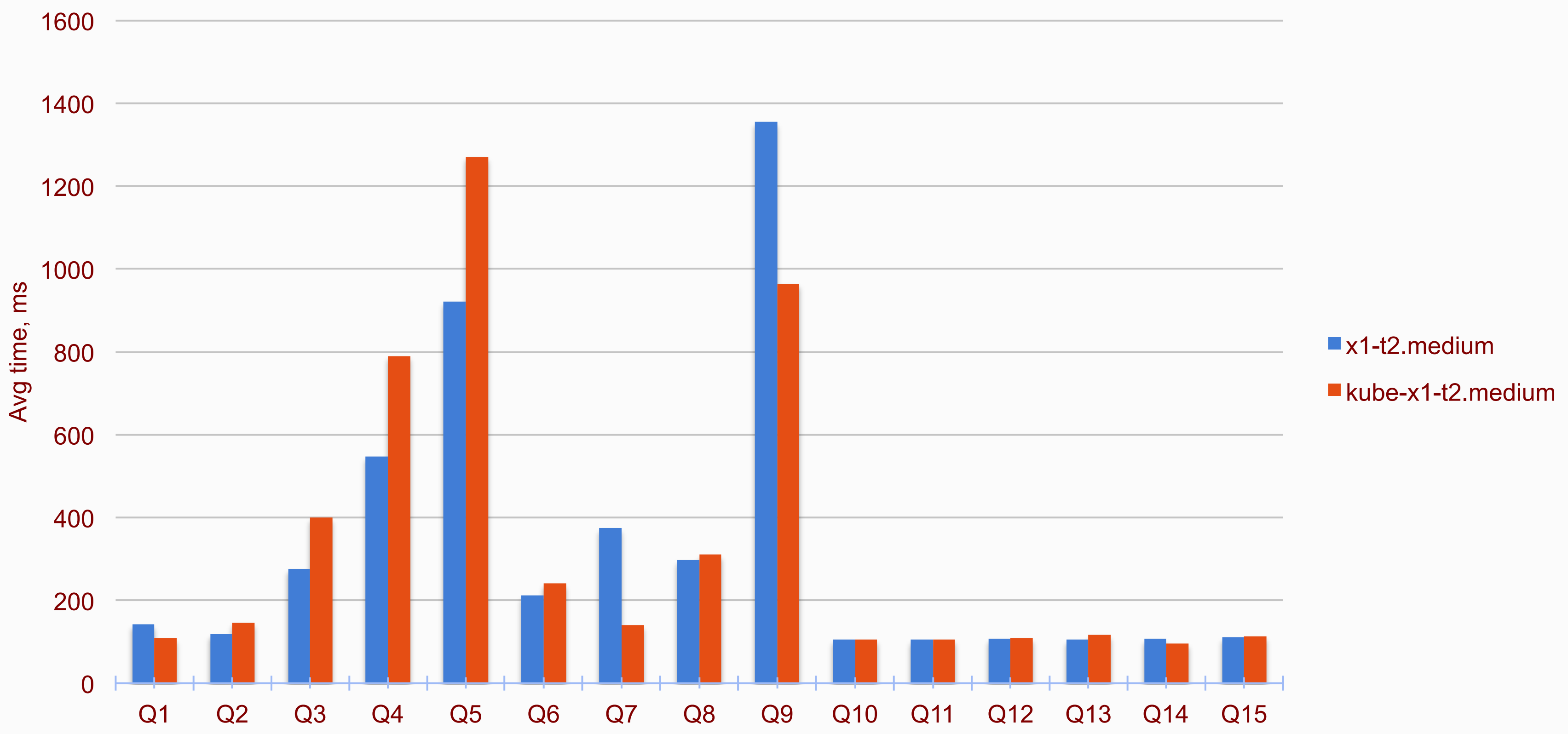
Operator = deployment + monitoring + operation



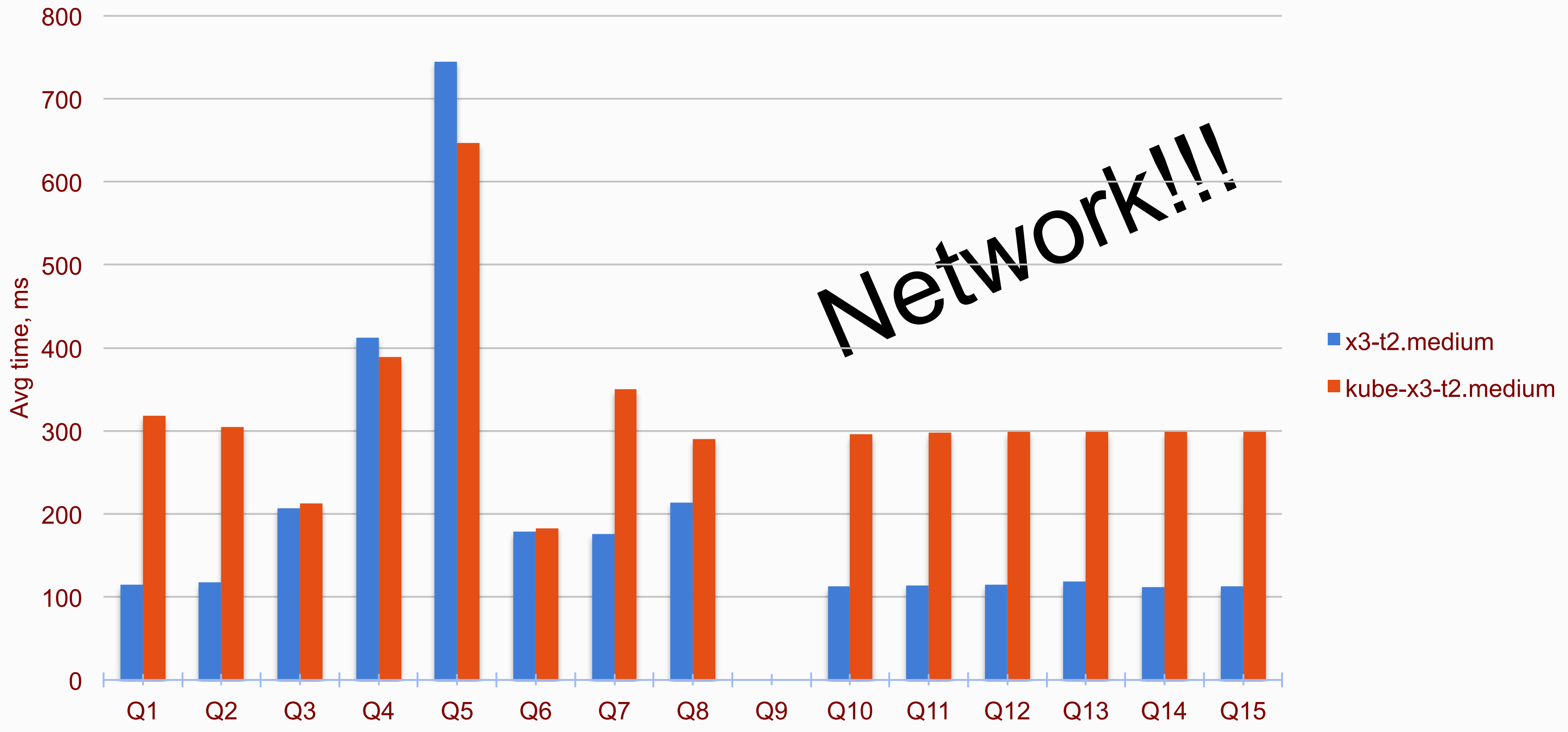
Codified Operational Knowledge

- Available already:
  - Monitoring to Prometheus / Grafana
  - Automatically create schema when adding shards/replicas
  - Track IP address changes when pod is re-provisioned
  - Rollback to the previous configuration in case of failure
- Planned / in-progress:
  - Configurable defaults
  - Multi-zone deployments
  - Advanced health checks
  - Automatic / semi-manual recovery
  - Replica re-provisioning
  - Storage management (tiered storage, backups)

Performance. AWS vs k8s@AWS. 1 node



Performance. AWS vs k8s@AWS. 3 nodes



Kubernetes Conainer Network Interface (CNI)

- Calico
- Canal (Flannel + Calico)
- flannel
- kopeio-vxlan
- kube-router
- romana
- Weave Net

To be tested!

Other Roadmap features

- Store history of configuration changes in a separate resource
- 'Canary testing’ of configuration changes at clean ClickHouse instance
- Operator status transparency
- Integration with ZooKeeper operator for automatic ZK provisioning
- Default configuration templates, including networking
- Integration with Altinity Cluster Manager

Advice, encouragement, and caveats

- Clickhouse operator is in beta
- Operator does not always detect errors in manifest files
- Error logging is limited, will be improved shortly
- Connectivity is a work in progress
- It’s a great way to explore cluster configurations
- Kubernetes is fun!

Please explore the operator and log issues on Github!!!

More information on Altinity ClickHouse Operator...

ClickHouse Operator Github Project:
https://github.com/Altinity/clickhouse-operator

Altinity Blog -- https://www.altinity.com/blog

Webinars -- https://www.altinity.com/webinarspage

Questions?

Thank you!

Contacts:
info@altinity.com

Visit us at:
https://www.altinity.com

Read Our Blog:
https://www.altinity.com/blog