
ClickHouse on Kubernetes!
Alexander Zaitsev
Altinity

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?

Other applications are already there

Easier to manage than deployment on hosts

Bring up data warehouses quickly

Portability

Is it easy to run ClickHouse on Kubernetes?

NOT REALLY

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

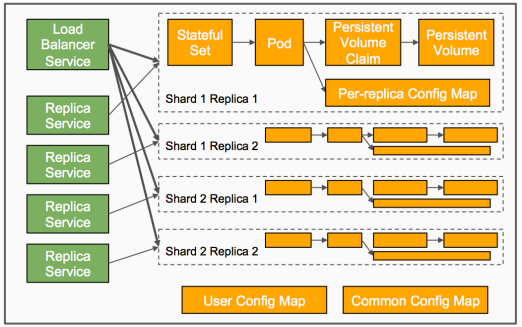
kubectl apply

ClickHouse Operator

create resources

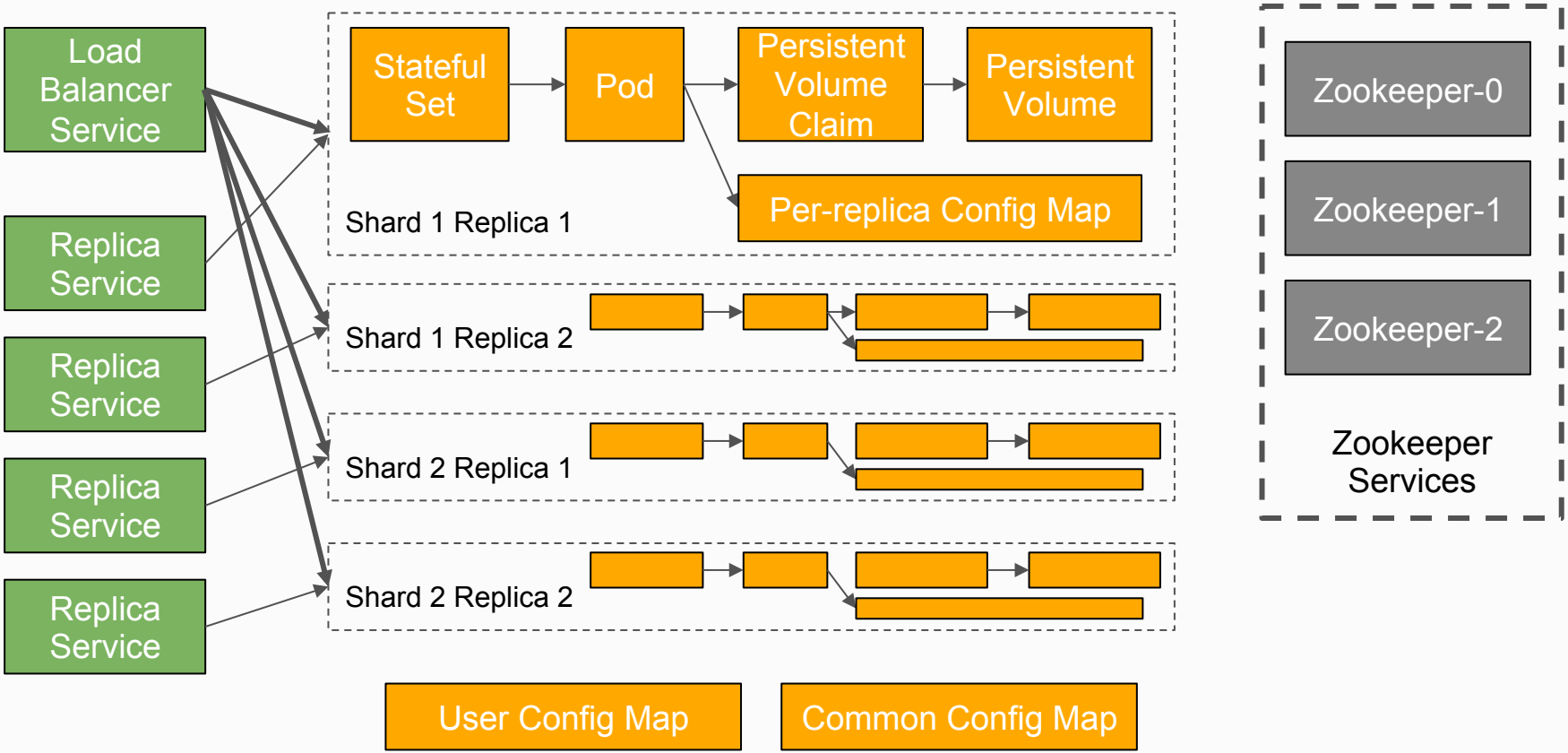
ClickHouseInstallation YAML file

(Apache 2.0 source, distributed as Docker image)



ClickHouse cluster resources

What does ClickHouse look like on Kubernetes?



Altinity ClickHouse Operator Quick Start

Installing 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"
        layout:
          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-demo-01-demo-0-0-0     1/1       Running   0          37s

NAME                             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/chi-demo-01-demo-0-0     ClusterIP      None            <none>        8123/TCP,9000/TCP,9009/TCP
37s
service/clickhouse-demo-01       LoadBalancer   10.98.143.187   <pending>     8123:32351/TCP,9000:32694/TCP
38s

NAME                                       DESIRED   CURRENT   AGE
statefulset.apps/chi-demo-01-demo-0-0      1         1         37s
```

Next let's add a shard

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

Next let's add a shard

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

$ kubectl get all -n demo

NAME                           READY     STATUS    RESTARTS   AGE
pod/chi-demo-01-demo-0-0-0     1/1       Running   0          2m
pod/chi-demo-01-demo-1-0-0     1/1       Running   0          33s

NAME                             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/chi-demo-01-demo-0-0     ClusterIP      None            <none>        8123/TCP,9000/TCP,9009/
TCP      2m
service/chi-demo-01-demo-1-0     ClusterIP      None            <none>        8123/TCP,9000/TCP,9009/
TCP      33s
service/clickhouse-demo-01       LoadBalancer   10.98.143.187   <pending>     8123:32351/TCP,
9000:32694/TCP   2m

NAME                                       DESIRED   CURRENT   AGE
statefulset.apps/chi-demo-01-demo-0-0      1         1         2m
statefulset.apps/chi-demo-01-demo-1-0      1         1         33s
```

How to access your ClickHouse data warehouse on Kubernetes

Connect from within Kubernetes using service DNS name

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

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 # optional
    clusters:
      - name: "demo-01"
        layout:
          shardsCount: 2
          replicasCount: 2
```

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

NOTE: Non-replicated tables are not converted to replicated automatically

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/password: secret
      demo/profile: default
      demo/networks/ip: "::/0"
    clusters:
      - name: "demo-01"
        layout:
          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:
    templates:
      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

Local volume mounts are also supported

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:
    templates:
      podTemplate: clickhouse-stable
      volumeClaimTemplate: storage-vc-template
  templates:
    podTemplates:
    - name: clickhouse-stable
      spec:
        containers:
        - name: clickhouse
          image: yandex/clickhouse-server:19.6.2.11
```

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
      spec:
        containers:
        - name: clickhouse
          image: yandex/clickhouse-server:19.6.2.11
            resources:
              requests:
                memory: "512Mi"
                cpu: "500m"
              limits:
                memory: "512Mi"
                cpu: "500m"
# Etc.
```

TIP: podTemplates may be applied at all levels:

- **installation**
- **cluster**
- **shard**
- **replica**

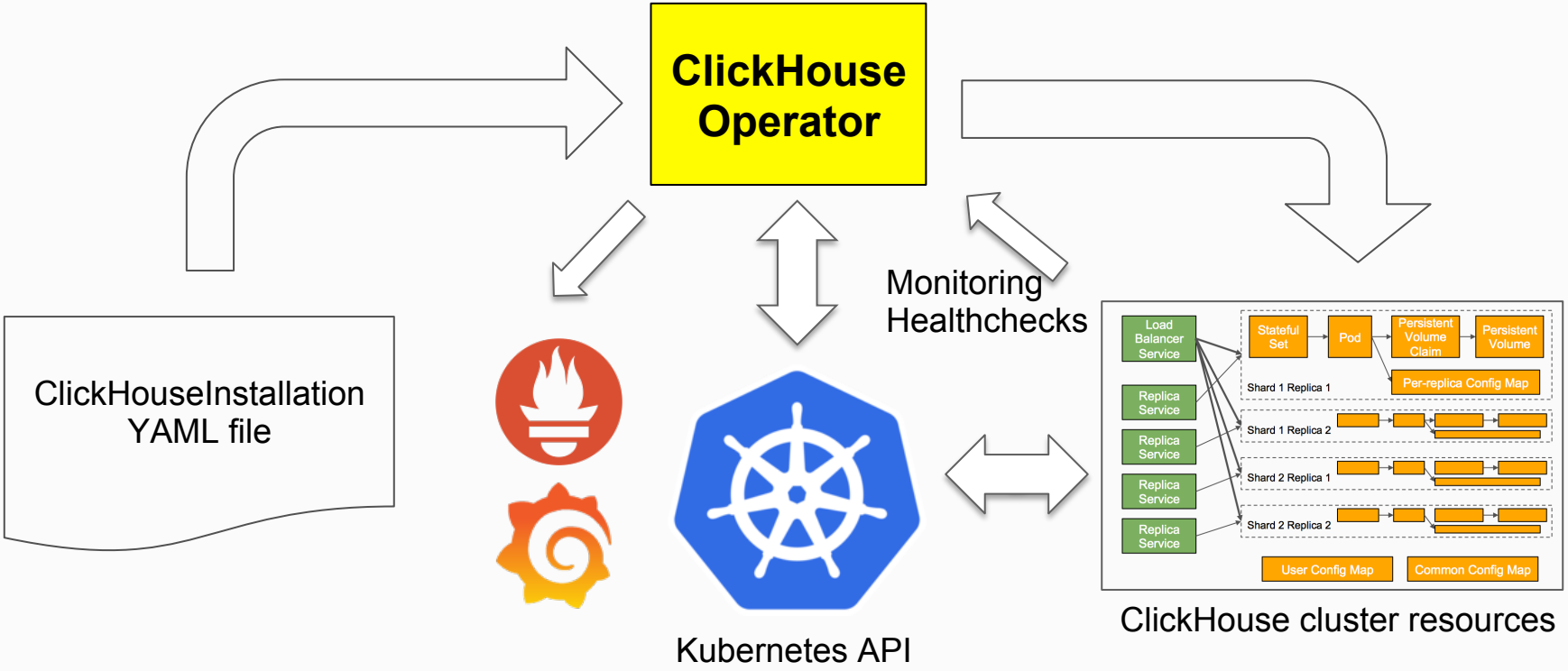
Other things to configure

- ClickHouse settings (profile, server settings), e.g.:

```
configuration:
  settings:
     compression/case/method: zstd
```

- Multi-zone deployment or other affinity rules
- AntiAffinity rules – e.g. every node is at the separate host
- Service types

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
  - Configurable defaults
- Planned / in-progress:
  - Advanced health checks
  - Jobs for:
    - Replica re-provisioning
    - Backups
    - Re-sharding
    - Global configuration changes

Operator Administration: version

```
# Get operator version

$ kubectl describe deployment clickhouse-operator -n kube-system
Name:                   clickhouse-operator
Namespace:              kube-system
CreationTimestamp:      Sat, 01 Jun 2019 23:44:46 +0300
Labels:                 app=clickhouse-operator
                        version=0.3.0

...
```

Operator can be upgraded using deployment image update

Operator Administration: configuration

```
# Get operator configuration

$ kubectl describe configmap etc-clickhouse-operator-files -n kube-system

Data
====
config.yaml:
----
# Namespaces where clickhouse-operator listens for events.
# Concurrently running operators should listen on different namespaces
# namespaces:
#   - dev
#   - test

################################################
##
## Additional Configuration Files Section
##
################################################

# Timeouts, retries, default user and so on
```

Operator Administration: ClickHouse defaults

```
$ kubectl describe configmap etc-clickhouse-operator-configd-files -n kube-system

Data
====
01-clickhouse-operator-listen.xml:
----
<yandex>
    <!-- Listen wildcard address to allow accepting connections from other containers and
host network. -->
    <listen_host>::</listen_host>
    <listen_host>0.0.0.0</listen_host>
    <listen_try>1</listen_try>
</yandex>

02-clickhouse-operator-logger.xml:
----
<yandex>
    <logger>
        <console>1</console>
    </logger>
</yandex>
```

Default config files are bundled in operator installation manifest

Other Roadmap features

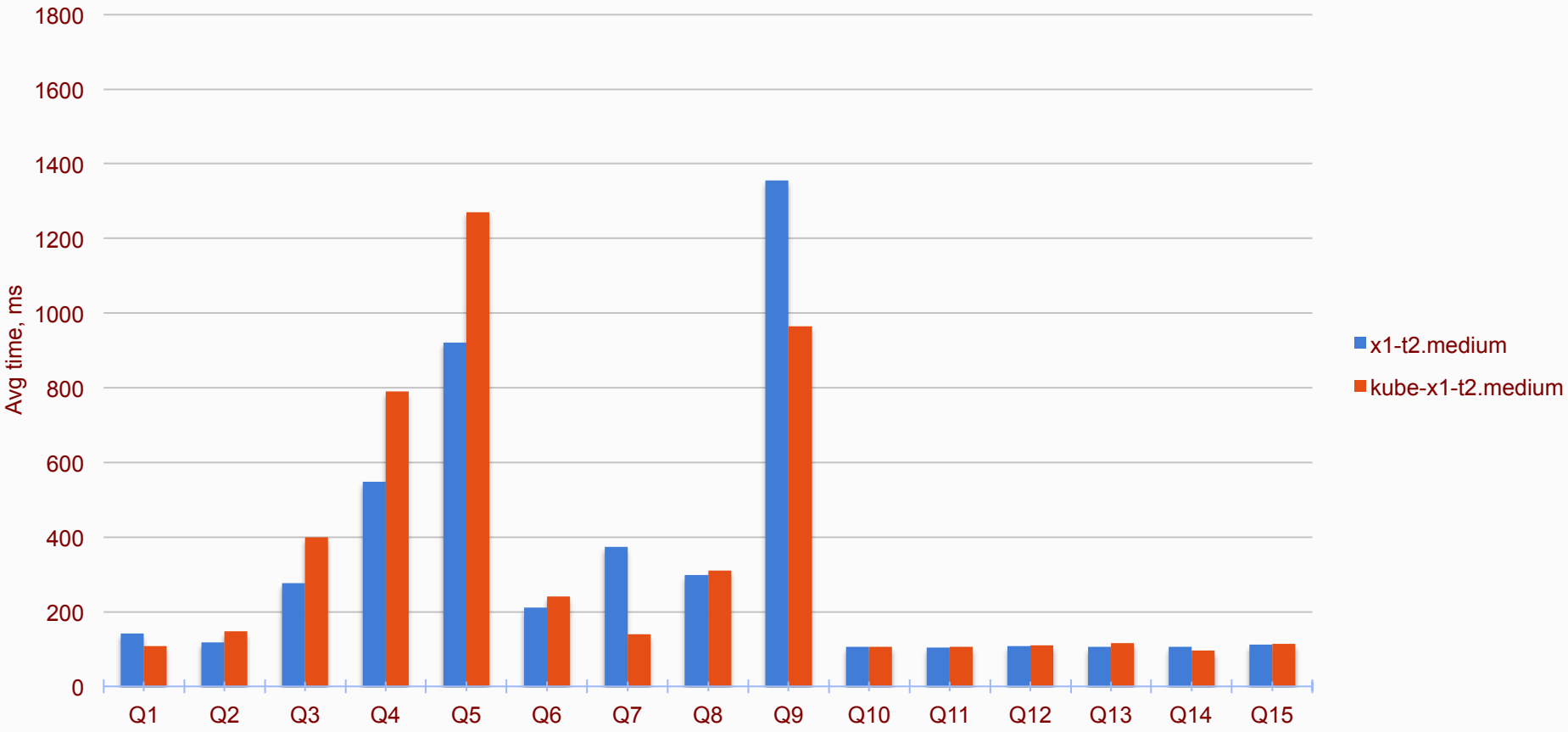
- Full support of Service spec
- Templates ‘inheritance’
- Store history of CRD changes in a separate resource
- 'Canary testing’ of CRD changes at clean ClickHouse instance
- Operator status transparency
- Integration with ZooKeeper operator for automatic ZK provisioning
- Integration with Altinity Cluster Manager

Does It Worth?

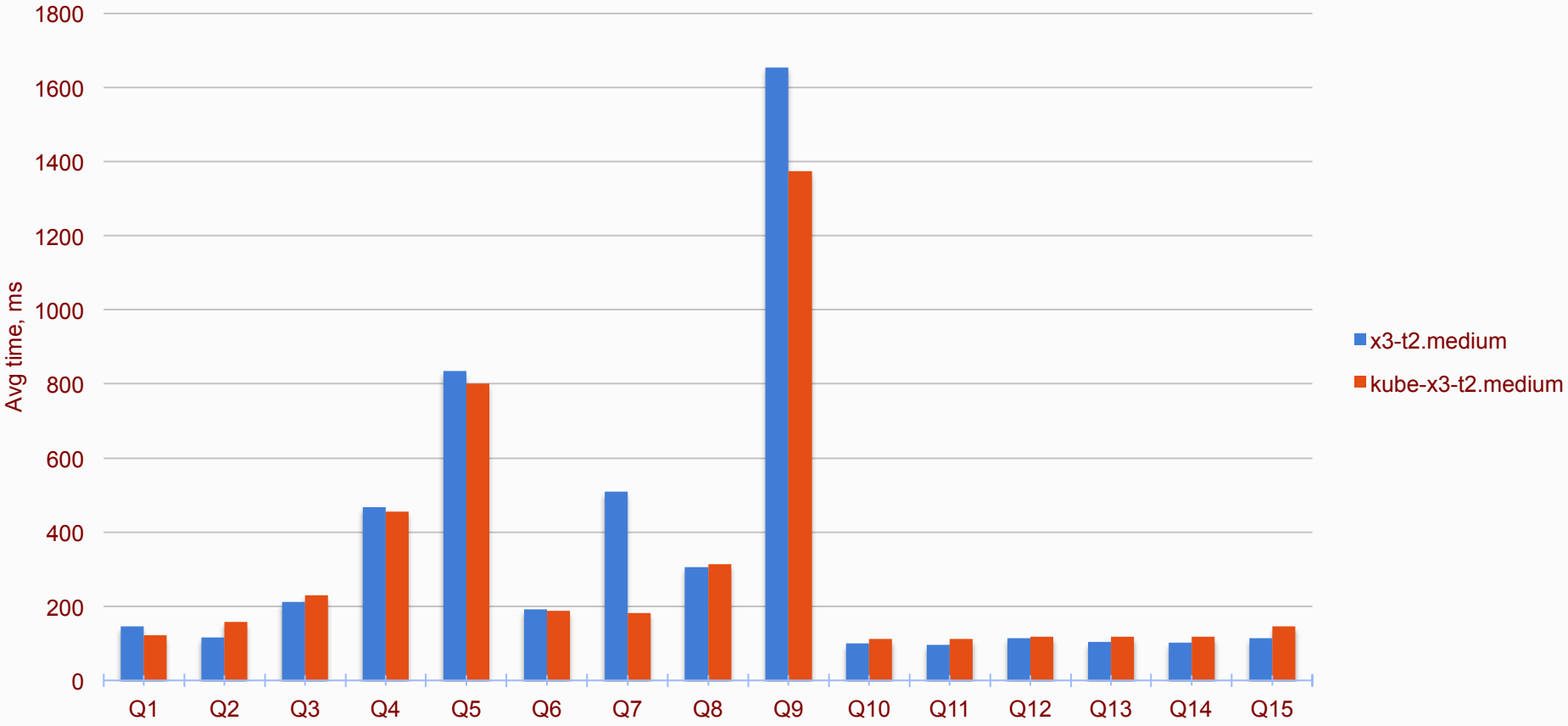
- Ease of configuration and deployment
- Ease of management
- Portability
- Built-in monitoring
- Operational knowledge

- What about performance?...

Performance. AWS vs k8s@AWS. 1 node



Performance. AWS vs k8s@AWS. 3 nodes



More information on Altinity ClickHouse Operator...

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

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

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