jampp

2020

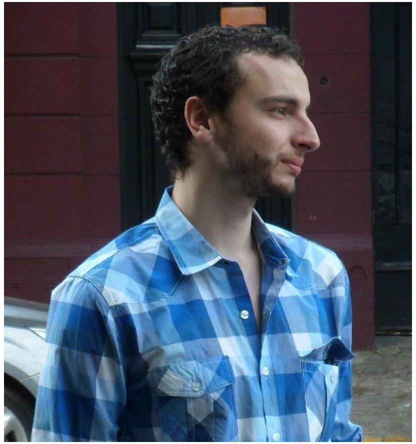
Introduction to Presto on Docker at scale

Federico Palladoro

About Me

Fede Palladoro

Devops & Data Infra Lead @ Jampp



@fedepalladoro

fede@jampp.com

jampp

Agenda

- Intro to Jampp data stack
- Previous Presto setup on EMR
- Migration to containers
- Orchestrators: Nomad vs Kubernetes
- Presto monitoring

What do we do at

Jampp?

#1

User Acquisition

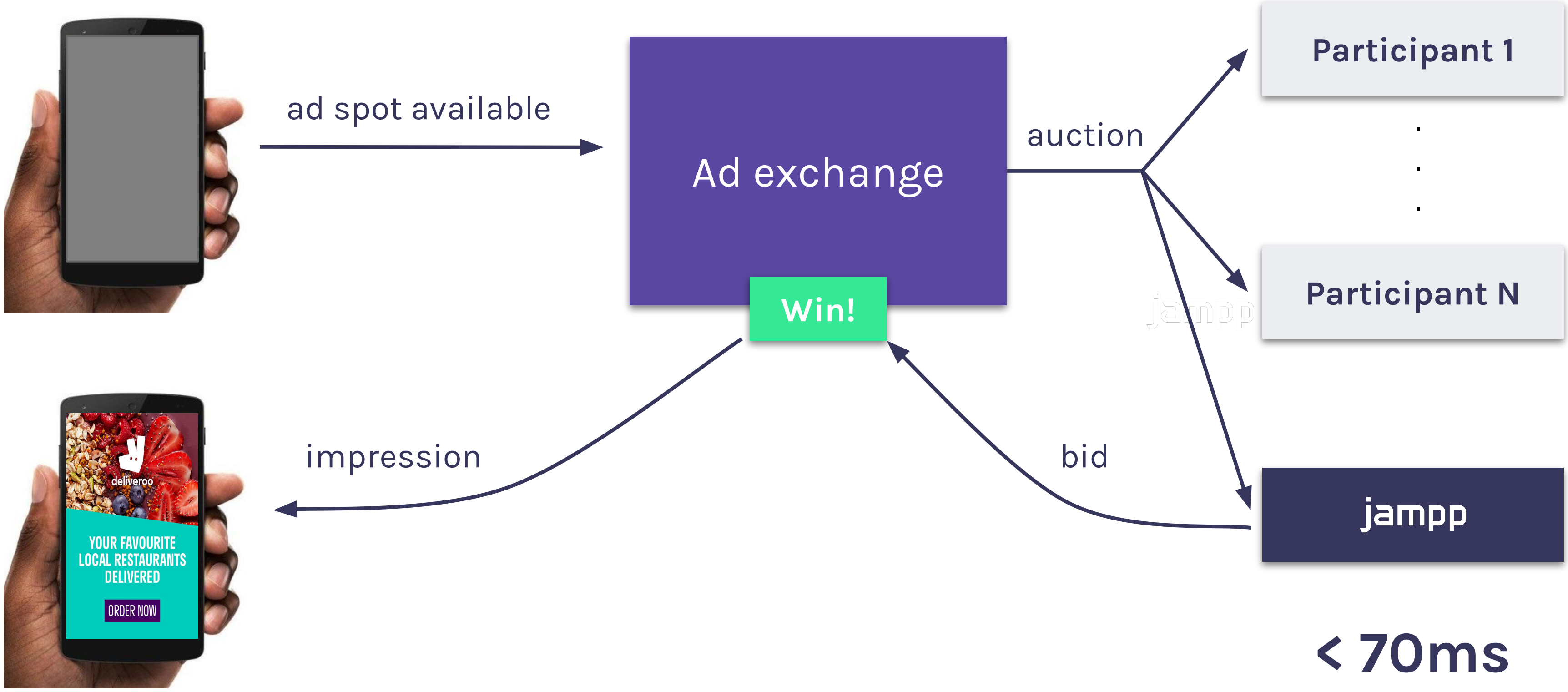
Find more people to install and use an app.

#2

App Retargeting

Re-engage existing users.

Real time bidding (RTB)



ad spot available

Ad exchange

auction

Participant 1

.
.
.

Participant N

jampp

bid

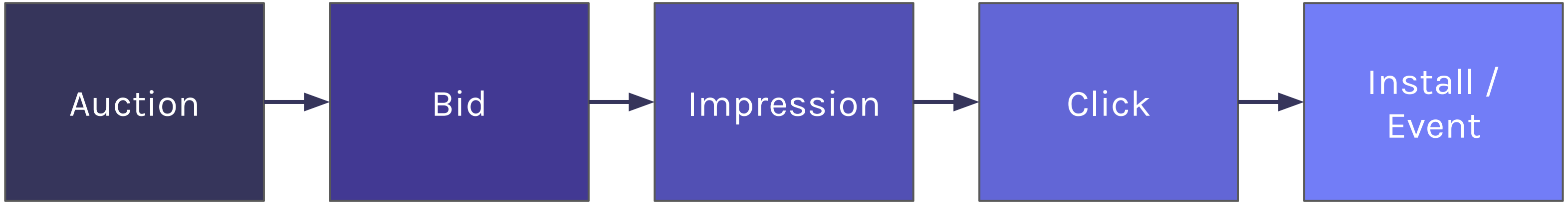
Win!

impression

< 70ms

Max Latency

Ad-Tech funnel



- Each step decreases volume by an order of magnitude
- The **data criticality increases** with each step.
- We can sample auctions to optimize costs but under no circumstances can we lose clicks, installs or events.
- Each table has **different access patterns** and needs a different partitioning scheme.

Some numbers

jampp

1M/s
Auctions received

+1.7 billion
Tracked events per day

150TB
Data processed by ELBs per day

+1000/h
Presto Queries

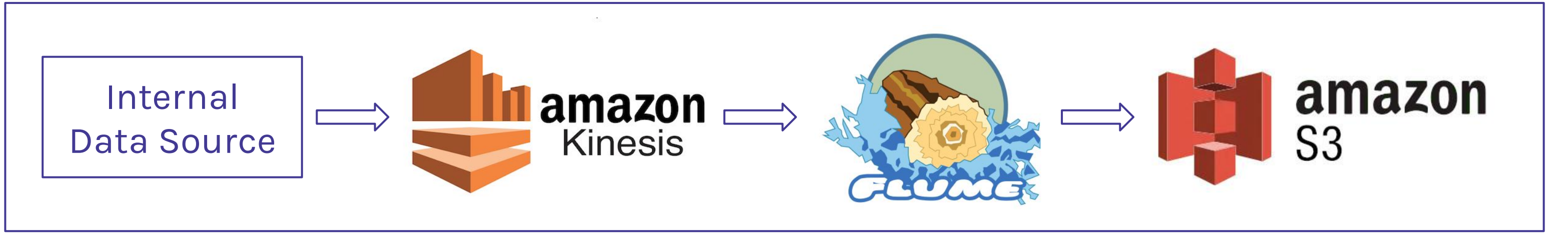
3
Presto Clusters

1,8TB-6TB
Total cluster memory

An overview of our

Data Infrastructure

Our pipeline operational unit



- One pipeline per event type.
- Focused on modularity and separation of concerns.
- Having them separated allows us to **optimize for cost without fear of losing critical messages**.

ETLs and data insertion

- Spark and Hive are very reliable for ETLs and insertion.
- We use the Hive Metastore as the main interface between engines.
- Airflow is an amazing tool for scheduling and orchestration.
- Storing data on S3 allows us to decouple compute from storage

Presto

Presto is the main interface with our Data Warehouse

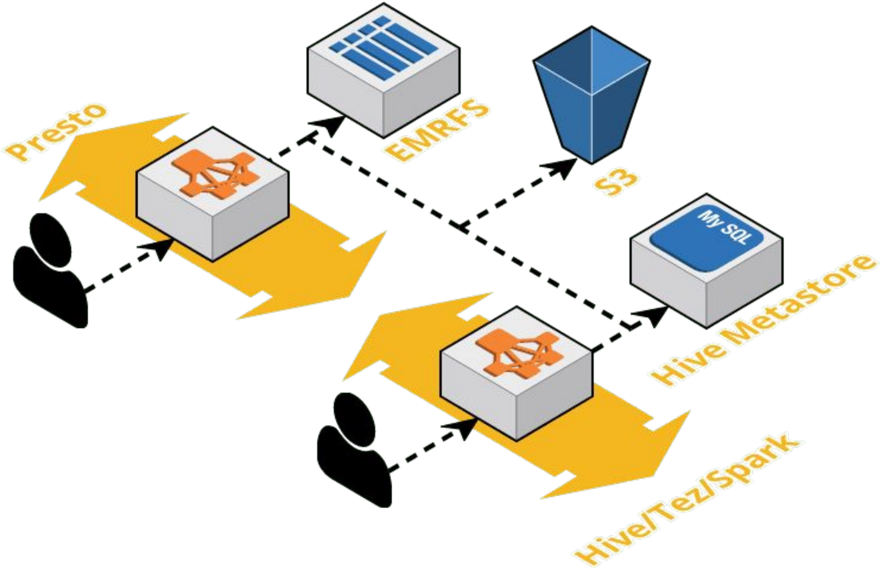
Through the years it became the main method of interacting with the Data Warehouse for every team in the company.

- Feeding our Machine Learning algorithms
- Building automatic audience segments
- Ad-Hoc queries through Apache Superset
- Templated reports through a custom UI
- Monitoring data quality

AWS EMR clusters

jampp

- 1 ETL cluster (Spark/Hive/Tez)
- 2 or 3 Presto clusters
- Data stored on S3, we don't use HDFS
- Each cluster is auto scalable depending the load
- Shared EMRFS on DynamoDB table
- Shared Hive Metastore on RDS



AWS Elastic MapReduce

The good

- Provisions out of the box many popular Big Data tools.
- Flexibility to tune applications and shape clusters as needed.
- Mainstream applications are frequently added to the app catalog, like PrestoSQL v338!

The bad

- Troublesome interaction between YARN (Hive, Spark) and non YARN apps (Presto).
- Low update frequency for fast pacing applications.
- Limited Presto support (i.e: no monitoring, no autoscaling on fleets)

The ugly

- They upgraded the OS to Amazon Linux 2 without EMR version change

Getting down to business

Moving Presto to containers

What?

- We decided to do two mayor changes:
  - Switch from PrestoDB to PrestoSQL
  - Take ownership of cluster provisioning and maintenance

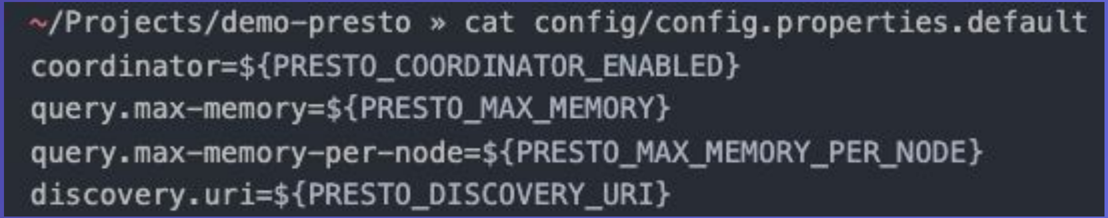
Why?

- Why PrestoSQL?
  - Community and user focused
  - Growing at a faster pace, more active contributors
  - Some known bugs already fixed (like hive bucketed tables)
  - Improved features like Cost Based Optimizer (CBO) and Security
- Why self-managed and Docker?
  - Lower costs (no EMR fees, no cluster overhead)
  - Quicker version upgrades
  - Local/ci environments just like prod/stg
  - Simpler configuration management

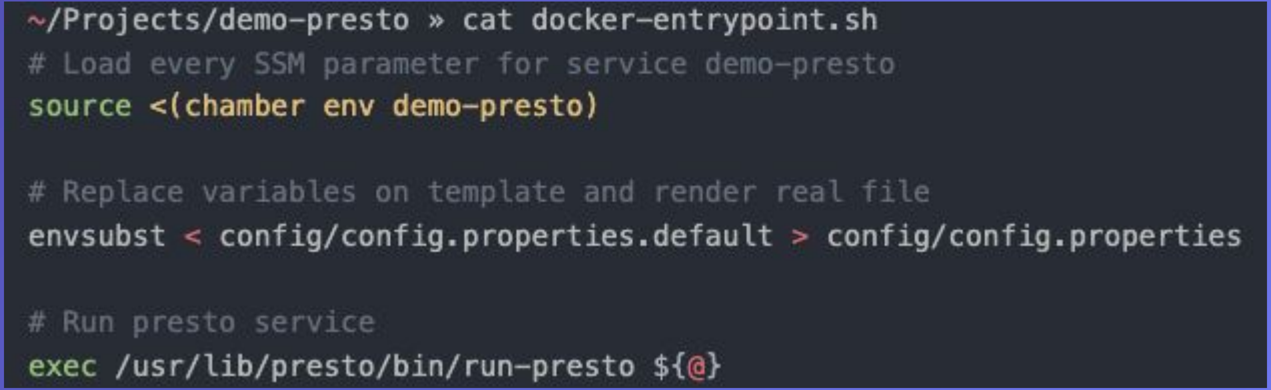
Building our docker image

- Based on the offical PrestoSQL image
- Dynamic configuration
  - Presto config and catalog files with templated values
  - Parameters and secrets stored on AWS SSM Parameter store
  - Segmentio's **`chamber`** to load parameters as env vars on runtime
  - Unix's envsubs to render final config files
- Additional tools like java agent for monitoring

Dynamic configuration



```
~/Projects/demo-presto » cat config/config.properties.default
coordinator=${PRESTO_COORDINATOR_ENABLED}
query.max-memory=${PRESTO_MAX_MEMORY}
query.max-memory-per-node=${PRESTO_MAX_MEMORY_PER_NODE}
discovery.uri=${PRESTO_DISCOVERY_URI}
```



```
~/Projects/demo-presto » cat docker-entrypoint.sh
# Load every SSM parameter for service demo-presto
source <(chamber env demo-presto)

# Replace variables on template and render real file
envsubst < config/config.properties.default > config/config.properties

# Run presto service
exec /usr/lib/presto/bin/run-presto ${@}
```

Orchestrator candidates

HashiCorp Nomad

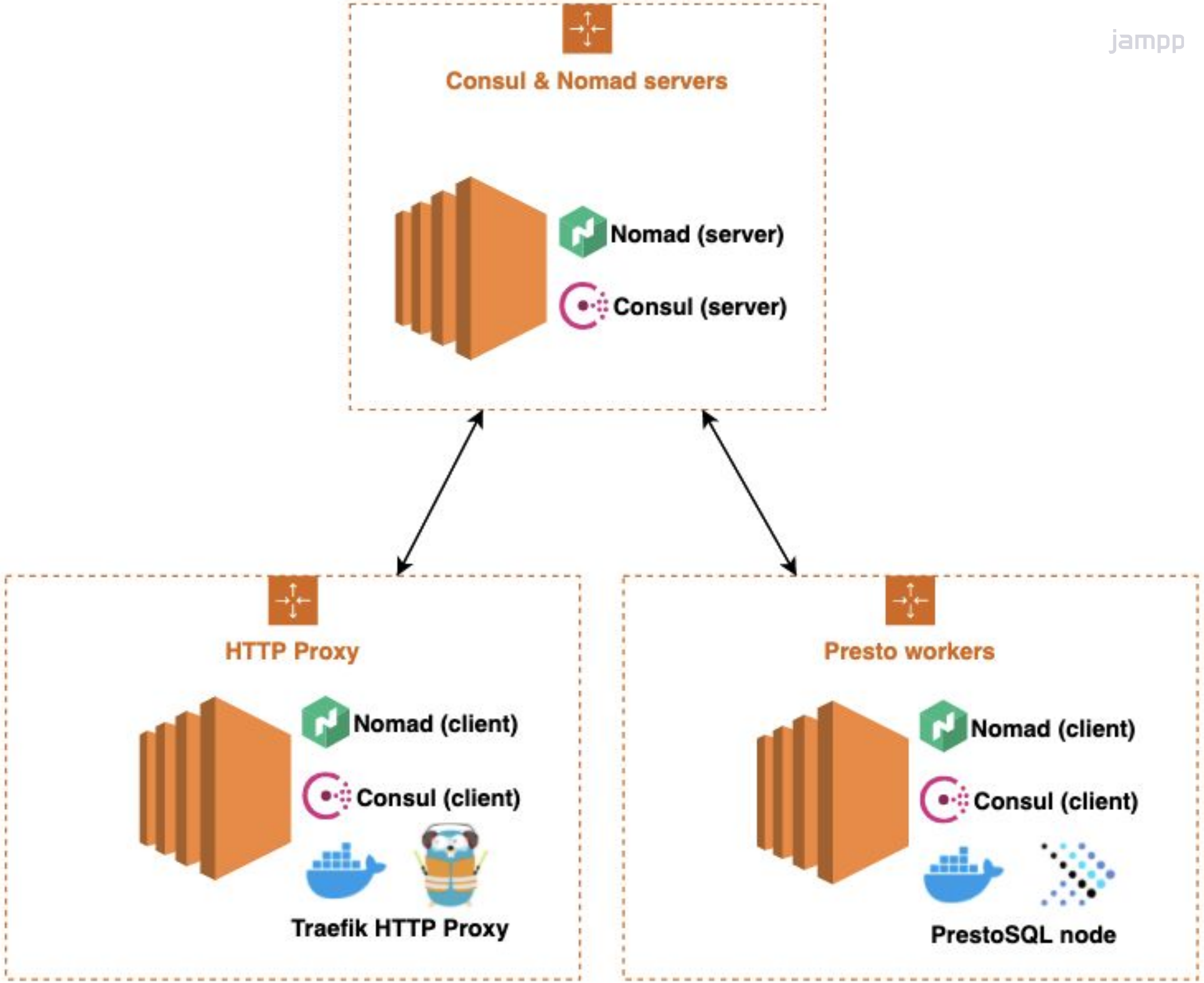
- The Tao of HashiCorp
- Orchestration with low complexity
- Support for non-container workloads
- Limited community - less known
- We already have it running

kubernetes

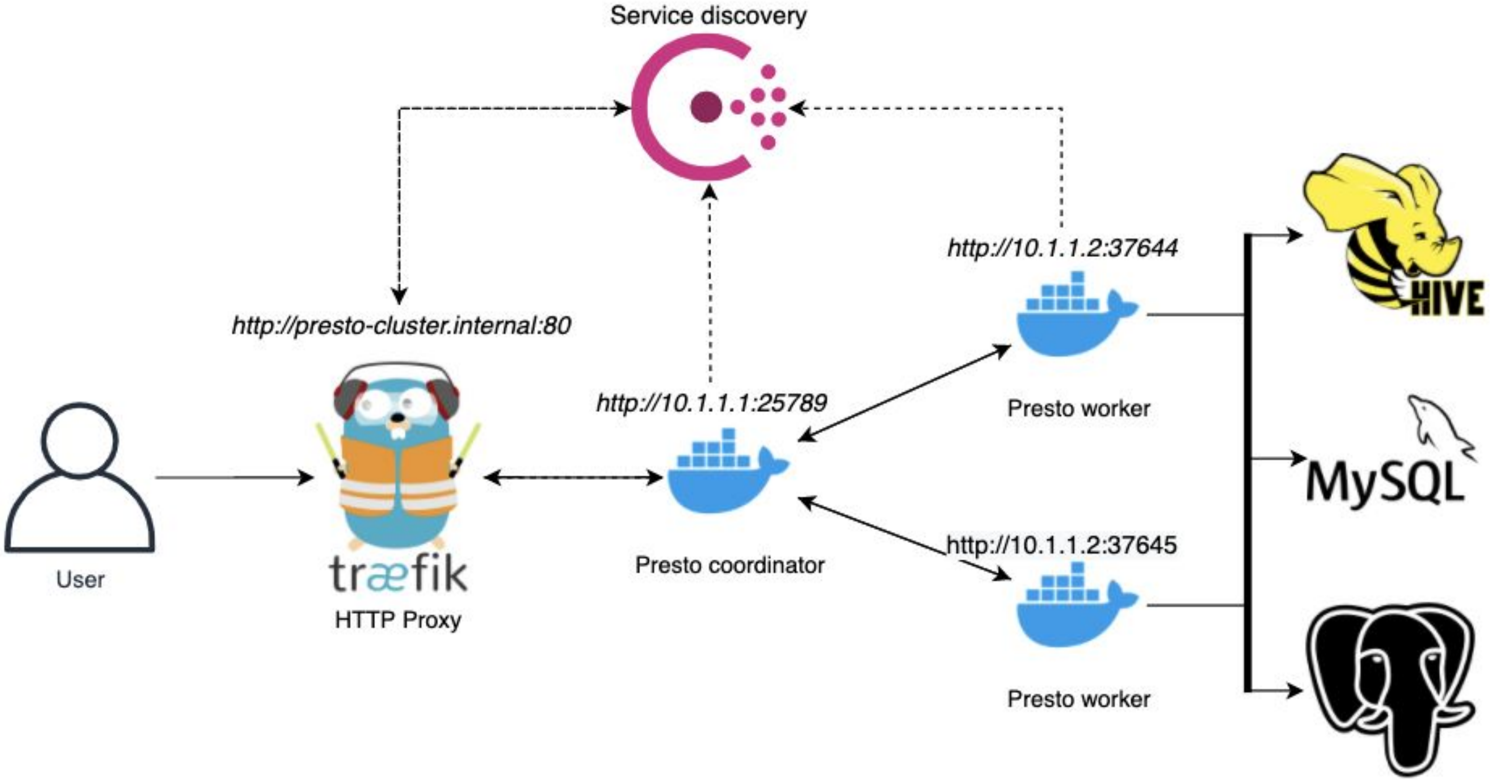
- Great community and tool ecosystem
- Industry-standard solution and battle tested
- High complexity, lot of internal “moving parts”
- Simple to spin-up using EKS/GKE/AKS

Presto setup on Nomad: Infra level

- Elastic autoscaling group for each component
- **Consul:** Service discovery + Distributed KV
- Control plane with Consul & Nomad
- Traefik as API Gateway / HTTP Proxy



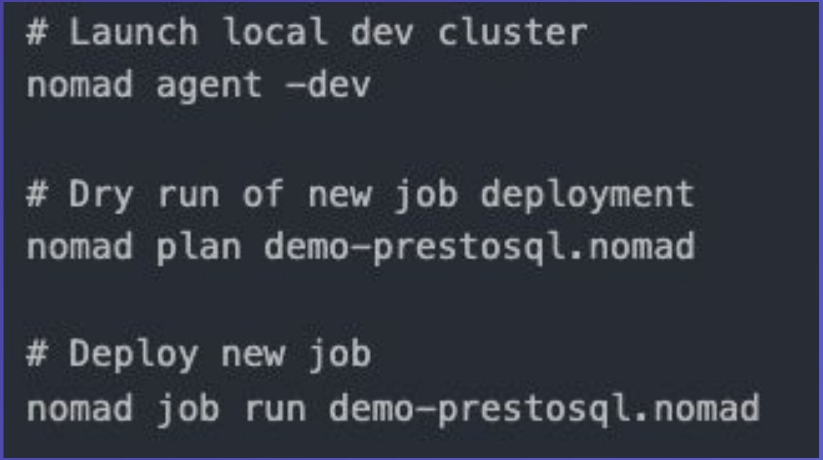
Presto setup on Nomad: App level



Extra Features

- Nomad job templating with Hashicorp Levant
  - Terraform-like workflow using a single template and a variable file per cluster/environment
- Autoscaling:
  - **Application level:** Nomad native support (CPU based)
  - **Cluster level:** Nomad official autoscaler agent
- Graceful scale-in of Presto workers
  - Autoscaling group hooks
  - Local node script
  - Put new status on presto node state endpoint `/v1/info/state`

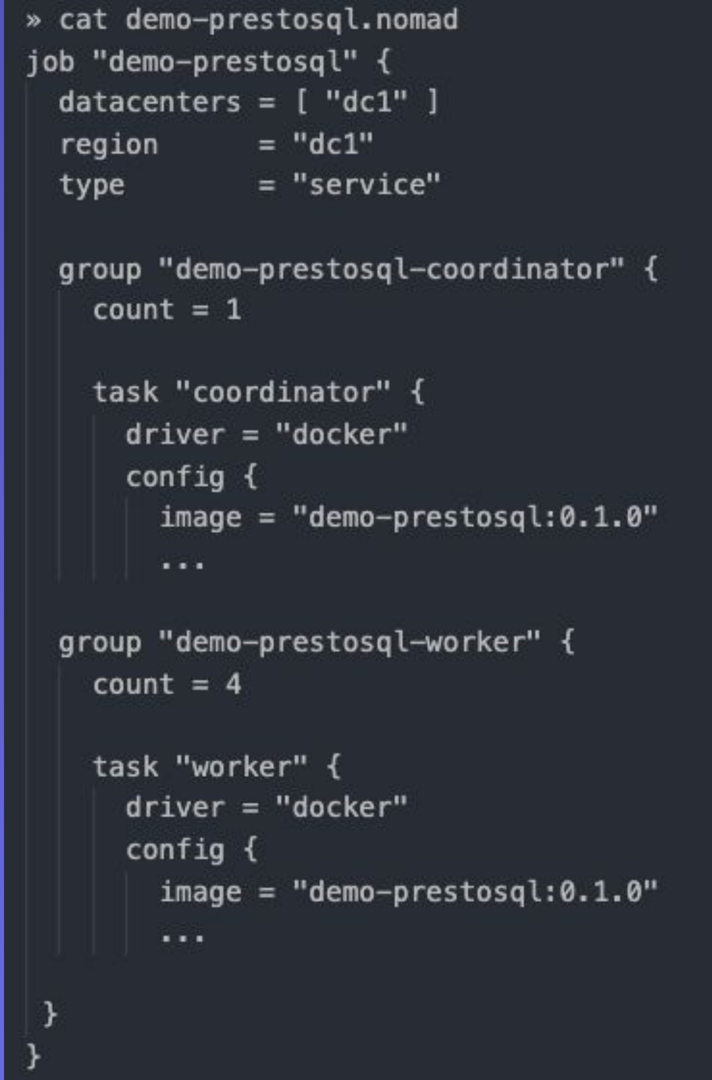
Local testing



```
# Launch local dev cluster
nomad agent -dev

# Dry run of new job deployment
nomad plan demo-prestosql.nomad

# Deploy new job
nomad job run demo-prestosql.nomad
```



```
» cat demo-prestosql.nomad
job "demo-prestosql" {
  datacenters = [ "dc1" ]
  region      = "dc1"
  type        = "service"

  group "demo-prestosql-coordinator" {
    count = 1

    task "coordinator" {
      driver = "docker"
      config {
        image = "demo-prestosql:0.1.0"
        ...

  group "demo-prestosql-worker" {
    count = 4

    task "worker" {
      driver = "docker"
      config {
        image = "demo-prestosql:0.1.0"
        ...

 }
}
```

Kubernetes

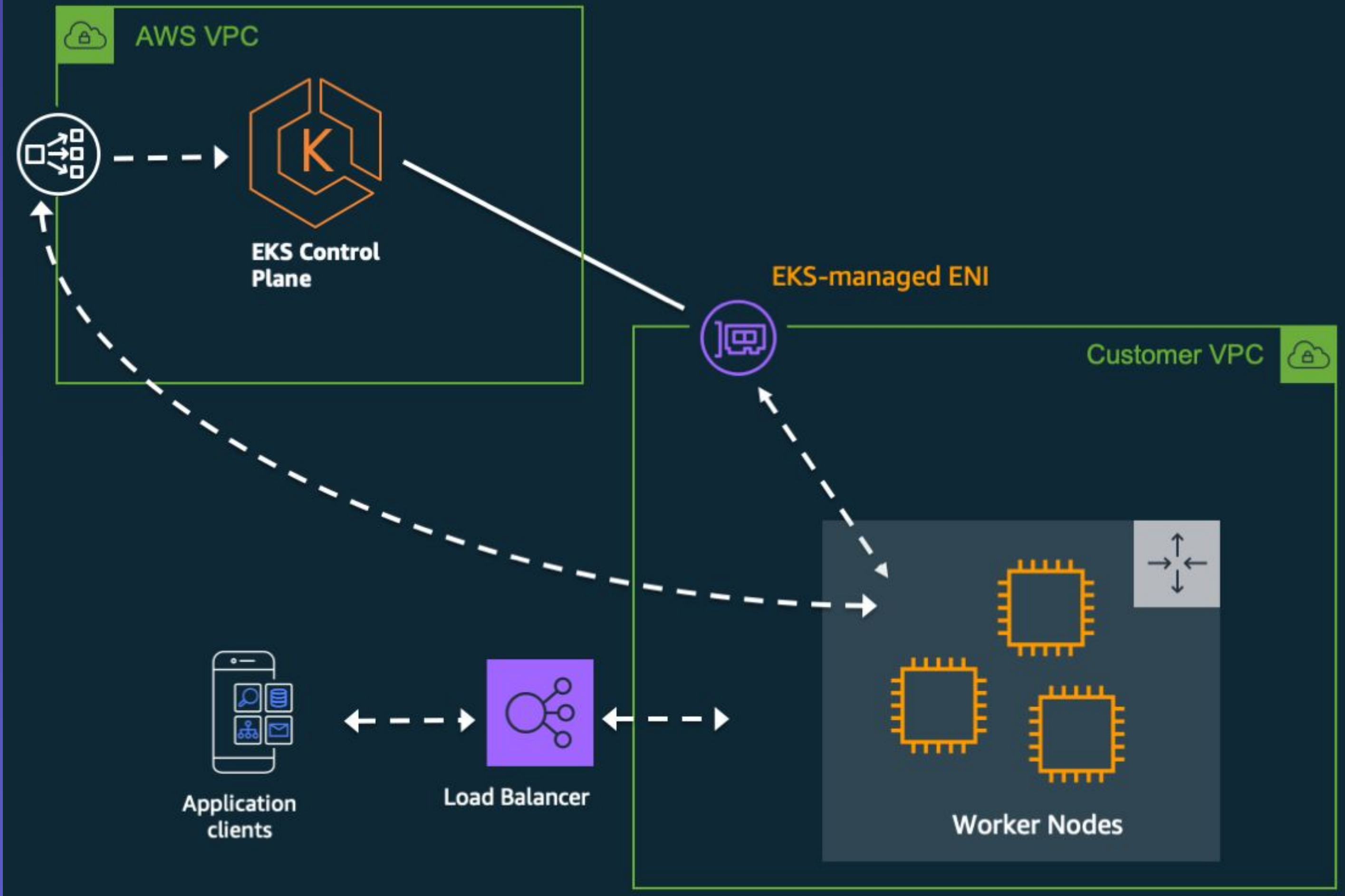
Helm charts

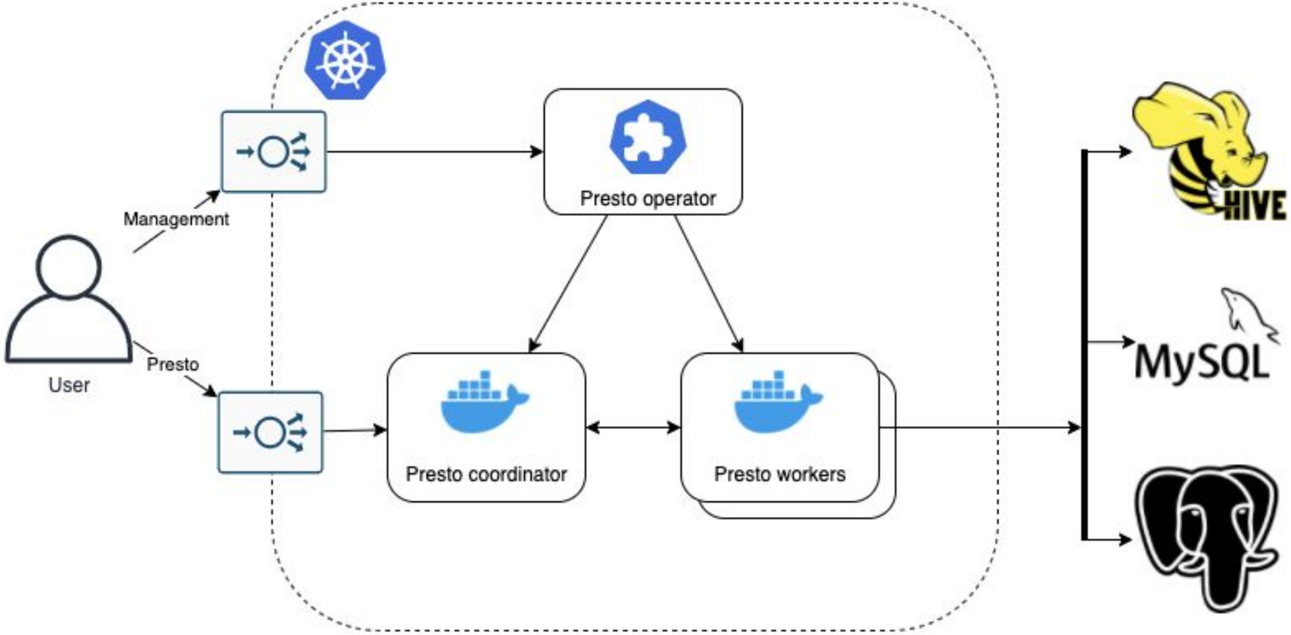
- Reusable templates of YAML artifacts
- Reduce duplicated code on multi-cluster environments
- Useful for resource creation/deployment (a.k.a Day 1)
- Presto on Helm:
  - PrestoSQL helm chart (non-official, open source)
  - Starburst helm chart (official, licenced/enterprise)

Operators

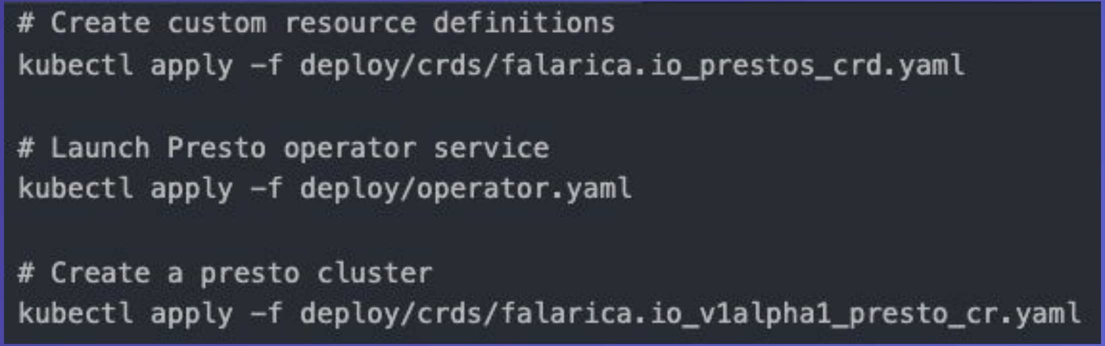
- Custom resource that extends k8s API
- Useful to ease maintenance on staful/complex workloads (a.k.a Day 2)
- Presto operators:
  - Falarica's presto operator (open source, just released)
  - Starburst presto operator (official, licenced/enterprise)

Kubernetes on AWS EKS





Presto on Kubernetes operator

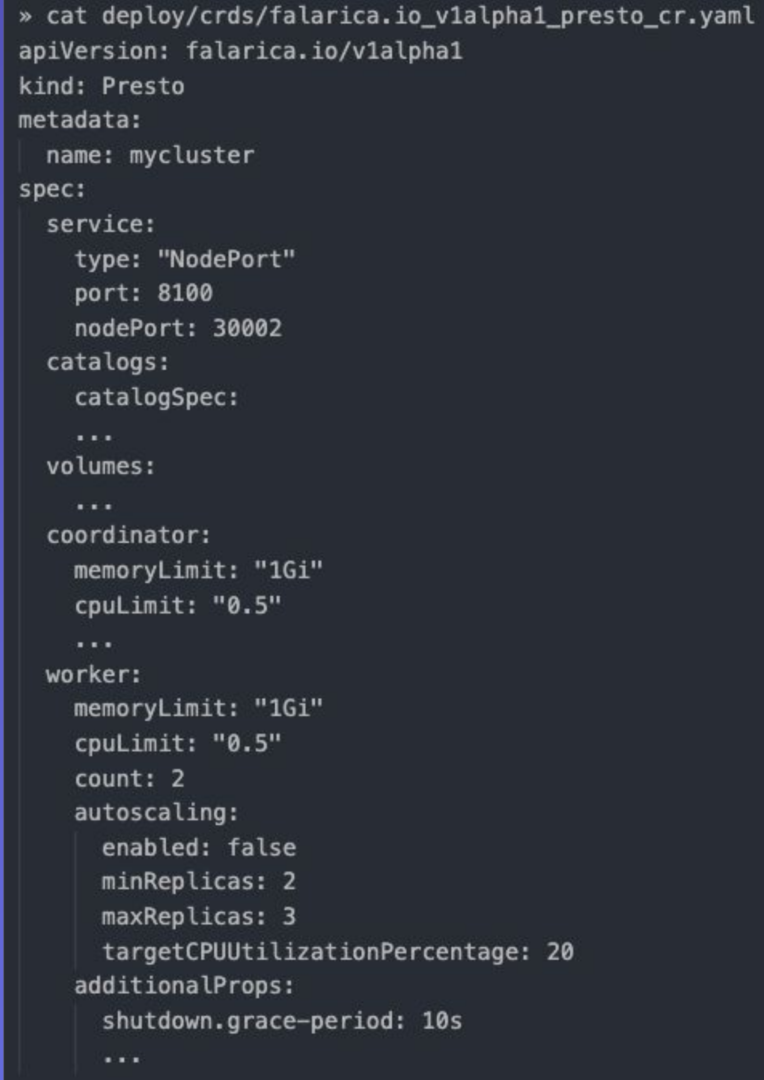


```
# Create custom resource definitions
kubectl apply -f deploy/crds/falarica.io_prestos_crd.yaml

# Launch Presto operator service
kubectl apply -f deploy/operator.yaml

# Create a presto cluster
kubectl apply -f deploy/crds/falarica.io_v1alpha1_presto_cr.yaml
```

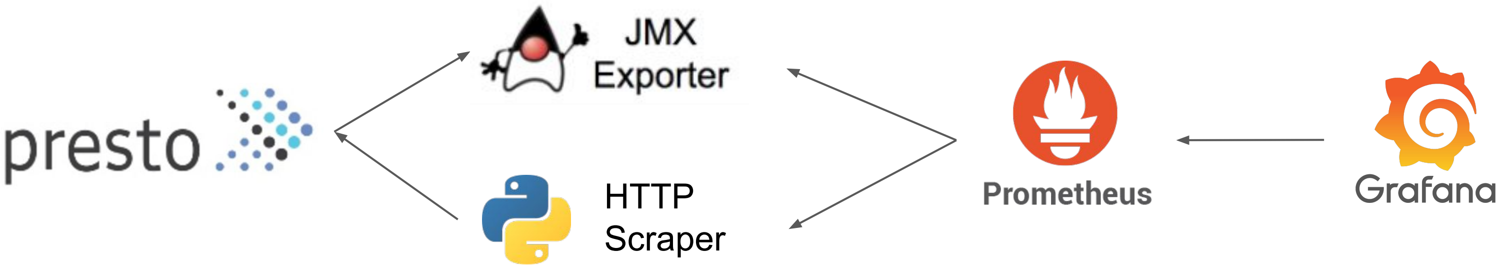
Presto on Kubernetes operator



```
» cat deploy/crds/falarica.io_v1alpha1_presto_cr.yaml
apiVersion: falarica.io/v1alpha1
kind: Presto
metadata:
  name: mycluster
spec:
  service:
    type: "NodePort"
    port: 8100
    nodePort: 30002
  catalogs:
    catalogSpec:
    ...
  volumes:
    ...
  coordinator:
    memoryLimit: "1Gi"
    cpuLimit: "0.5"
    ...
  worker:
    memoryLimit: "1Gi"
    cpuLimit: "0.5"
    count: 2
    autoscaling:
      enabled: false
      minReplicas: 2
      maxReplicas: 3
      targetCPUUtilizationPercentage: 20
    additionalProps:
      shutdown.grace-period: 10s
      ...
```

Monitoring stack

- We expose low level metrics with JMX java agent for Prometheus.
- Developed a custom exporter to get user level usage metrics from `/v1/query` endpoint
- Prometheus stack collects mbeans attributes.
- Grafana for dashboards and custom searches.



Monitoring relevant metrics

Low level (JMX)

- Memory pools, Heap usage.
- Garbage collection frequency and duration.
- Cluster size and nodes status.
- Active, Pending and Blocked queries.

User level (HTTP API)

- Finished, canceled and failed queries per user.
- Normalized query analytics to detect usage patterns.

Next steps

- **Leverage CBO** to improve query performance.
- Evaluate the usage of a **Presto gateway** to manage query routing to multiple clusters.
- Enable autoscaling from Prometheus metrics.
- Define SLI’s and SLO’s to measure reliability.
- Evaluate Presto on k8s + AWS Fargate (serverless containers)

- **Segment.io chamber:** https://github.com/segmentio/chamber
- **The Tao of Hashicorp:** https://www.hashicorp.com/tao-of-hashicorp
- **Nomad tutorial:** https://learn.hashicorp.com/tutorials/nomad/get-started-install
- **PrestoSQL helm chart:** https://hub.helm.sh/charts/stable/presto/0.2.1
- **Starburst helm chart:** https://docs.starburstdata.com/latest/k8s/overview.html
- **Falarica's presto operator:** https://github.com/falarica/steerd-presto-operator
- **Starburst presto operator:** https://docs.starburstdata.com/latest/kubernetes/overview.html
- **AWS EKS Architecture:** https://aws.amazon.com/quickstart/architecture/amazon-eks/

Link references

Thanks!!

jampp

We are hiring!!
jampp.com/jobs

@fedepalladoro

fede@jampp.com