Optimizing Trino using Spot Instances

- Rupesh Kumar Perugu
- Santhosh Venkataraman

Who we are

BI Platform Team
@Zillow

Rupesh Kumar Perugu
Senior Software Engineer,
Data Platforms

Santhosh Venkataraman
Software Engineer, Data Platforms

Agenda

- What is Zillow?
- Trino at Zillow
  - Our uses cases and scale
  - Query Infra Overview
- Operating Trino
  - Previous State
  - Concerns with Previous State
- Optimizations
  - Spot Instances with Ocean(Spot IO)
  - Spot Percentage Tuning
  - Impact measurement
- Future Work
- Thank you

About Zillow

- Reimagining real estate to make it easier to unlock life's next chapter
- Offer Customers an on-demand experience for selling, buying, renting and financing with transparency and nearly seamless end-to-end service
- Most visited real estate website in the United States
  - 234 million average monthly unique users

Our Brands

Zillow®

Zillow PREMIER AGENT™

Zillow HomeLoans

Zillow ClosingServices

Zillow Rentals

trulia

out east

StreetEasy

hotpads.

Our Real Estate Software

ShowingTime+

dotloop.

BRIDGE INTERACTIVE™

ShowingTime™

MORTECH®

What we do

- **What we do as BI Platform team at Zillow?**
  - Enable access to data and metrics in datalake in an efficient, secure, self-serving and performant manner

- **Who are our clients?**
  - Analytical users at Zillow like
    - Data scientists, Data analysts, Product Managers, Data engineers and BI engineers

- **Use Cases:**
  - Scheduled reports to generate metrics to unlock opportunities for Zillow
  - Adhoc analysis across various domains (like PA, ZHL, Rentals, etc.)

- **What we use?**
  - We use Trino, distributed SQL query engine as a query layer for users to interact with data at scale.

Trino at Zillow

Scale(Per Day):

👥 ~ 1250

- Avg queries ~ 65K
- Avg read ~ 600 TB
- Peak memory usage ~ 2.5 TB
- Avg execution time ~ 250 Hr
- Avg P95 time ~ 20s

Trino at Zillow

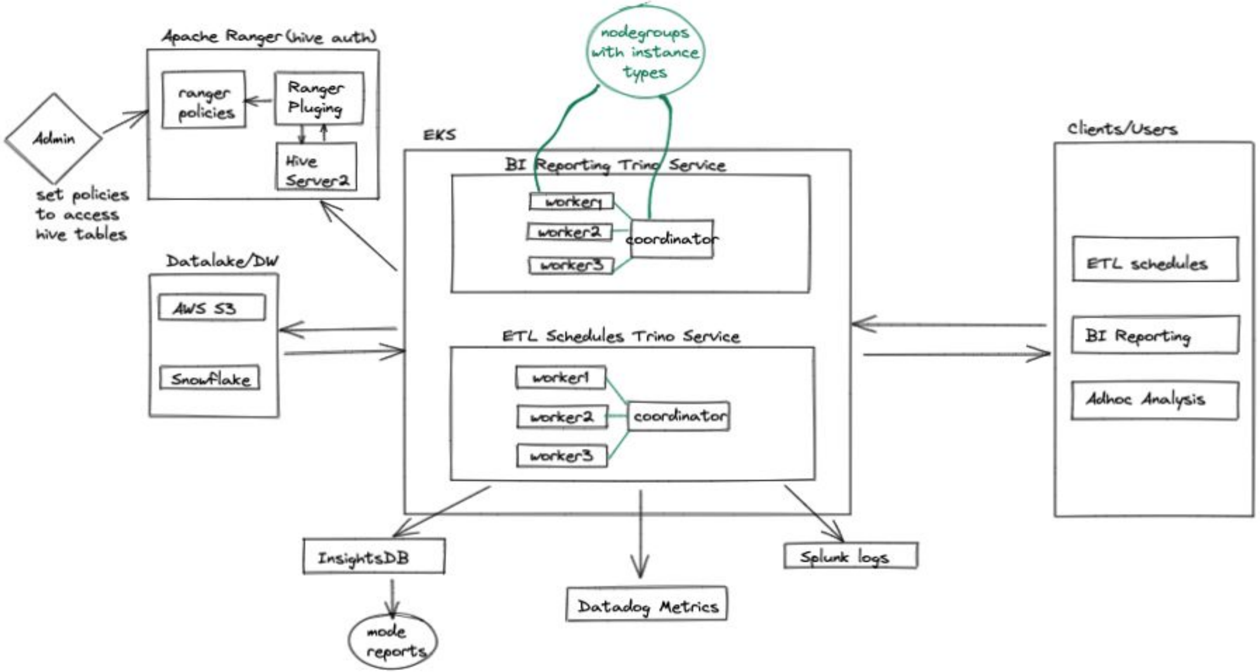
- We have about **6 Trino services** in live that share the load across Zillow based on BI Reporting, ETL schedules, adhoc analysis and Visualization services.

- Each service has **8 min workers** and will scale up to **60 max workers** based on HPA (Horizontal Pod Autoscaler) and CPU Utilization(>70%).

- All Trino services are hosted on Elastic Kubernetes Service (EKS, managed service within AWS)

- Instance Types
  - On-Demand Instances
  - M5a (16xlarge)
    - 64gb vCPUs
    - 256 GB memory

Query Infra Overview



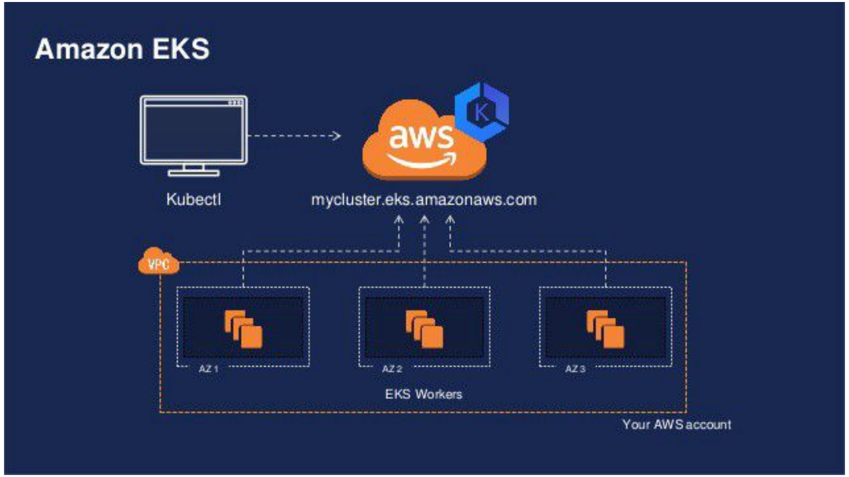
Operating Trino (Previous State)

- With EKS you can either use AWS Managed nodes(aws managed) or Worker Groups(self managed) for node provisioning and life cycle management.

- *Worker groups* provided more flexibility on choosing AMI and deploying EKS nodes to AWS local zones which improves latency

- Various approaches of choosing instance types on aws -
  - Spot Instances (no spot ratio)
    - 90% of less cost compared to on-demand but less reliability
  - On-Demand Instances
    - More expensive

AWS Worker Groups (Self Managed)



Concerns with Previous State

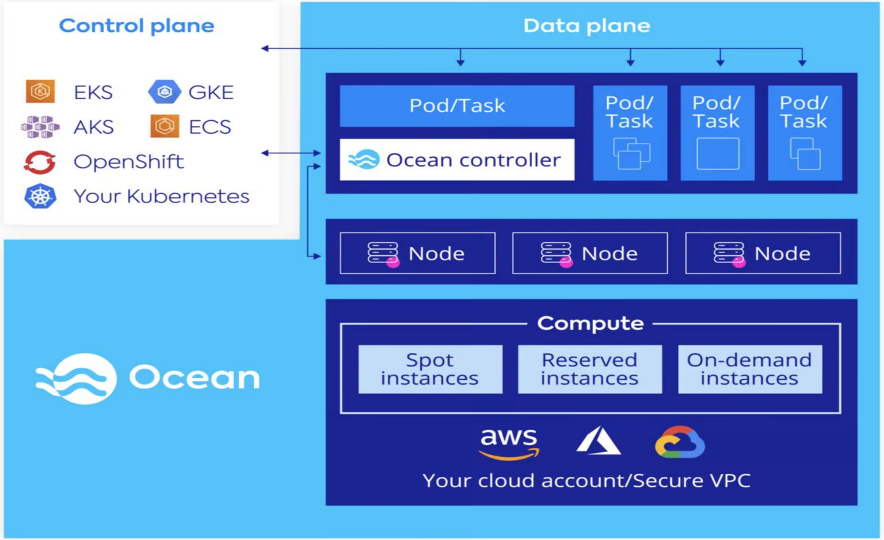
- Soon we started seeing challenges using worker groups very early in transition with
  - ***Spot Interruptions***
  - ***Cost creep***
  - Nodes running out of capacity in the region and
  - Consistent challenges of keeping nodes upto date was a lot of overhead for the team.

- Attempts to solve above concerns in terms of cost and reliability -
  - Selecting ***mix of spot instance types*** from different regions based on the availability
  - ***Exploring spot ratio*** which isn't available using worker groups on AWS.
  - Replacing all worker nodes with On-Demand instance
  - We found provision of nodes via ***Ocean(Spot IO)*** on EKS which seemed to be a good fit for our issues.

Optimizations (using Spot Instances)

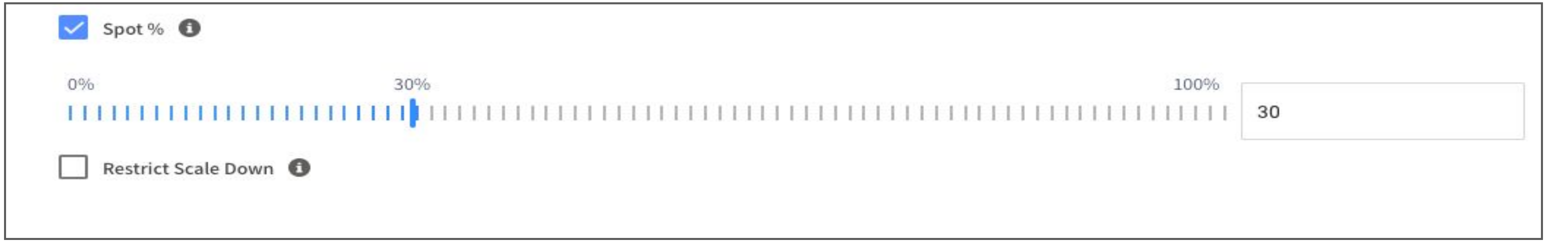
Ocean with Spot.io provides a cost-effective way of running workloads by effectively managing spot instances. We want to understand the advantages and limitations for Trino workloads in these areas using Spot IO -

- Flexibility on ***Spot Ratio*** Selection
- Flexibility on ***Node Type Selection***
- ***Spot Interruptions Management***
- AutoScaling
- ***Fallback to On-Demand Instances***
- Availability Zones (AZ) placements



Optimizations (Spot Percentage Tuning)

- Spot Percentage Tuning
  - Why?
    - To reduce cost without interrupting reliability of service
  - How?
    - Trial and Error
      - 50% spot usage made the cluster less reliable due to frequent spot instance interruptions
      - 10% didn't give the optimized cost benefit we were looking for.
      - Finally, 30% spot selection with fall back to on-demand when not available improved our cost without sacrificing reliability much better than other spot percentages for our load.



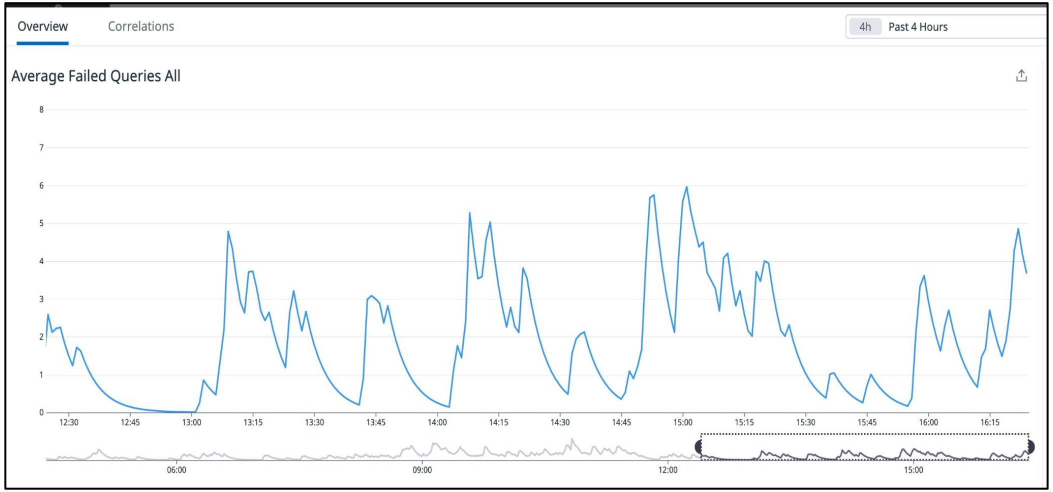
☑ Spot %

30

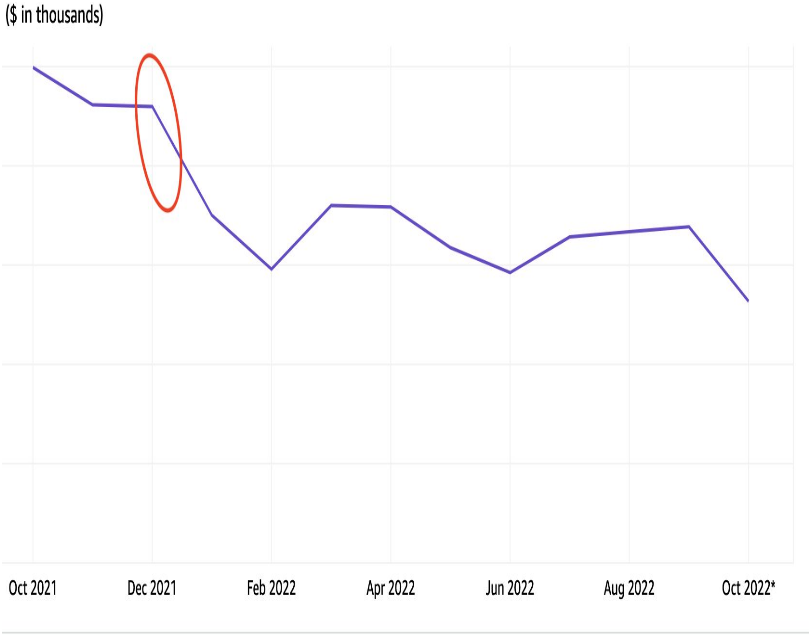
□ Restrict Scale Down

Optimization (Spot Percentage Tuning)

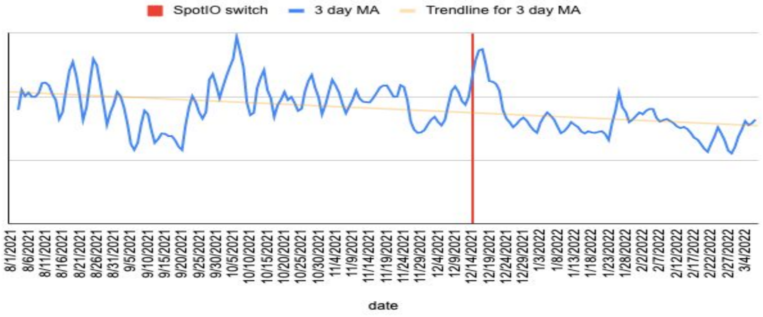
- Spot Percentage Tuning **as we scale**



Optimizations (Impact Measurement)



- After rollout of spot instance usage and spot percentage tuning, savings of ***25% per year*** is observed without sacrificing price and reliability



Future Work

- **Fault Tolerance Execution**
  - **Retry for failed queries** to improve user reliability and experience without sacrificing cost and performance.

- **Optimize Instance Types**
  - Selecting right set of instance types based on the computation, memory, network transfer etc. for different use cases.
  - Exploring Graviton Instances (M6g, R6g and C6g types) being one of the option among others which are 40% better at price without sacrificing the performance.

- **Increase Query Performance**
  - Create **index types**(like Bitmap, dictionary etc) across datalake to improve query speed by atleast 7x times.
  - **Smart Caching** for un-optimized datalakes and frequently used top tables

Questions?

Thank You

https://career.zillowgroup.com/careers