AppDeveloperCon

NORTH AMERICA

AppDeveloperCon NORTH AMERICA

Searching for Serverless Candidates in the Jungle of Microservices

Yash Bhatnagar

Why Serverless

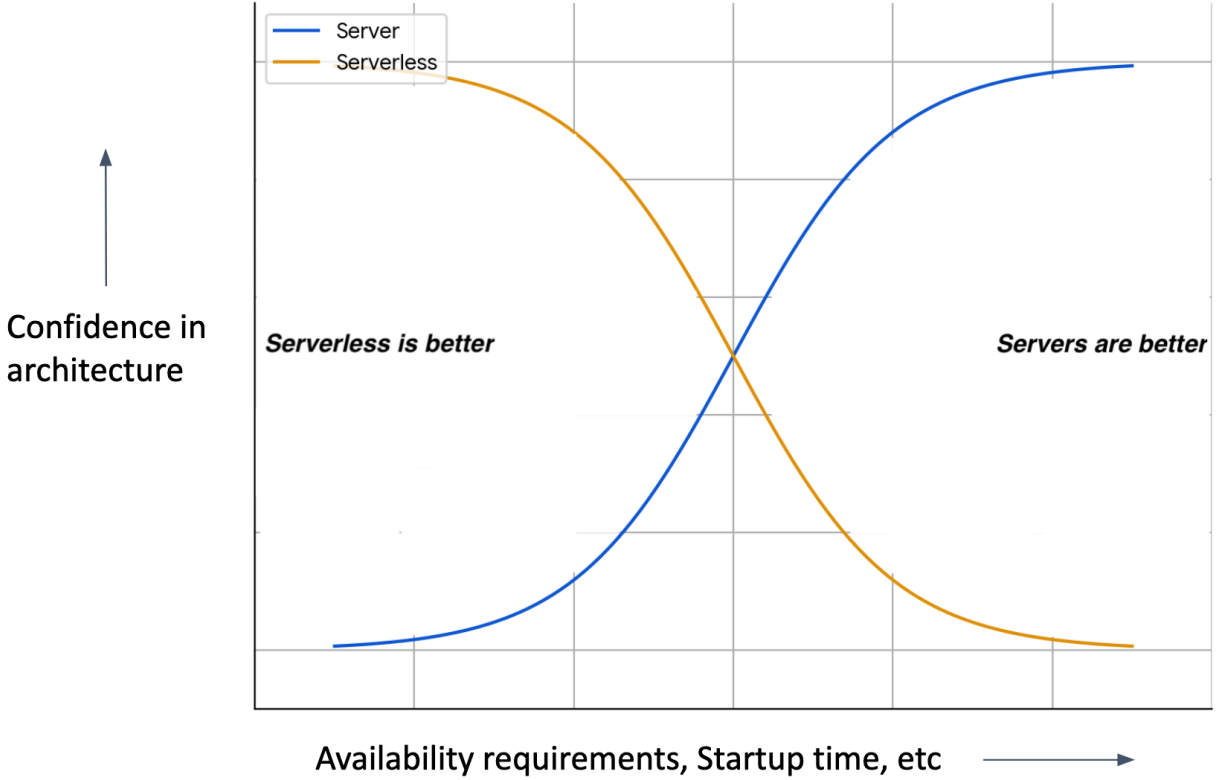
- Principle - Develop core functionalities as independent functions / jobs
  - invoked and orchestrated as required
- Limits idle time and resources
  - E.g: No need to run a full fledged microservice for a batch job
- Helps save significantly on infra costs
- Kubernetes - Jobs, CronJobs, etc.

Which to choose

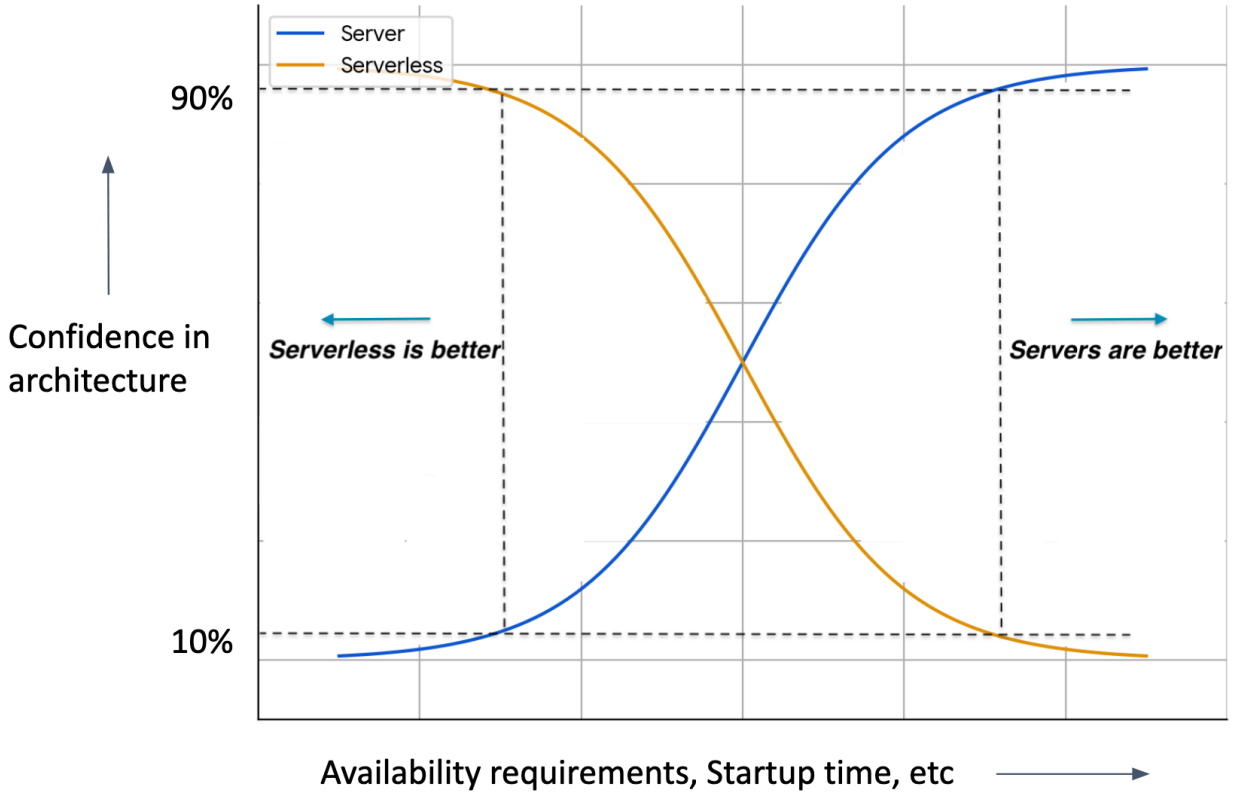
- Microservice vs Serverless - the architect's decision

Microservices better suited	Serverless better option
○ Complex logic, ○ tight integration(s) and dependencies with other services / systems ○ high availability and boot time etc.	○ Low coupled, independent workload ○ periodic / seasonal workloads ○ bursty workloads - frequent scale changes etc.

A simple Qualitative model



The area of uncertainty



But why the uncertainty?

- Because afterall, we are predicting!!
  - Customers and their usage patterns, etc
  - Future scalability / roadmap changes - e.g: other services needing to leverage this service
  - High availability requirements.

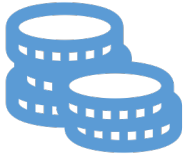
Result - Serverless adoption is lesser

Unsure patterns – generic microservices are safer bet

Risk aversive approach-optimization comes later

Time – Most valuable resource

Fast paced CI-CD models



Lack of insights - lack of motivation

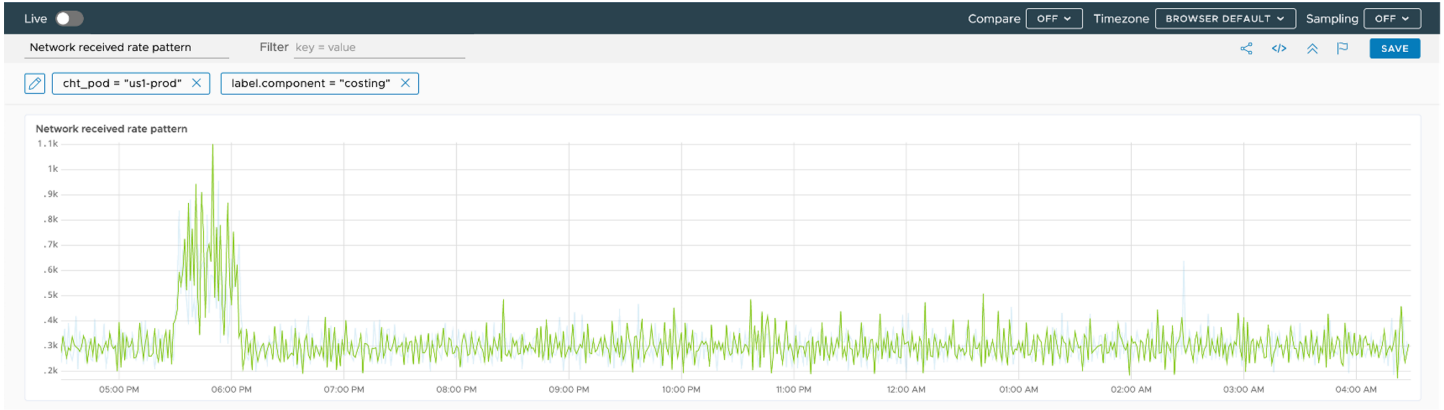
Cost Savings??

Performance improvement???

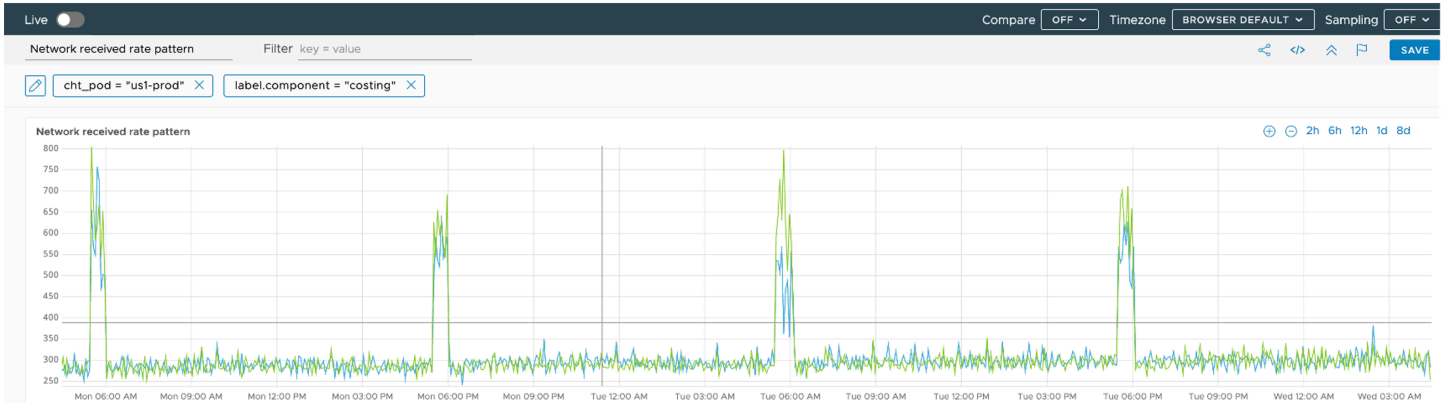
So, how to fix this?

- Lets take the simple obvious case - a microservice that is effectively performing event-driven batch processing.

The evidence



Spiked utilization



Seasonal utilization

But these patterns…

Not easy to detect.

- Vary with time.
- Differ across resources.
- Dependent on customer usage patterns
- Have Co-relation with several applications.
- Differ across time granularity.

How to find the best candidates – real case

- This was the our use case – how to detect any such candidates, without knowing anything about the services

How do we go about doing it?

AppDeveloperCon NORTH AMERICA

- Leverage the application data - use metrics

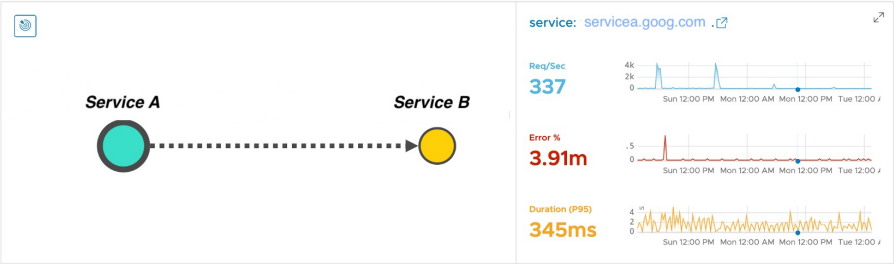
Metric	T n	T n-1	T n-2	Tn-3	…
vCPU (mC)	1.332	1.233	1.172	1.142	…
Memory usage	2.44 GB	2.44 GB	2.10 GB	2.22 GB	…
Network Tx	4.5k	3.12k	200	120	…
Network Rx (KB/s)	5.989 k	8.187k	660	980	…
Ephemeral Storage usage rate	298 KB/s	10 ms	10 ms	9 ms	
…	…	…	…	…	…

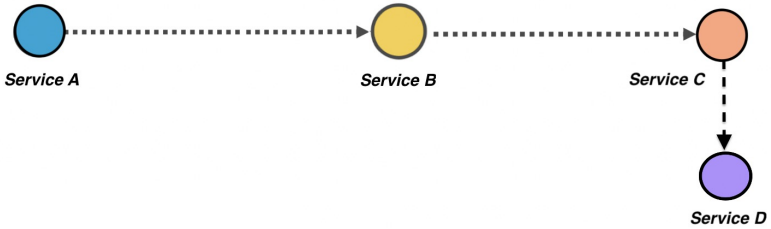
How do we go about doing it?

AppDeveloperCon NORTH AMERICA

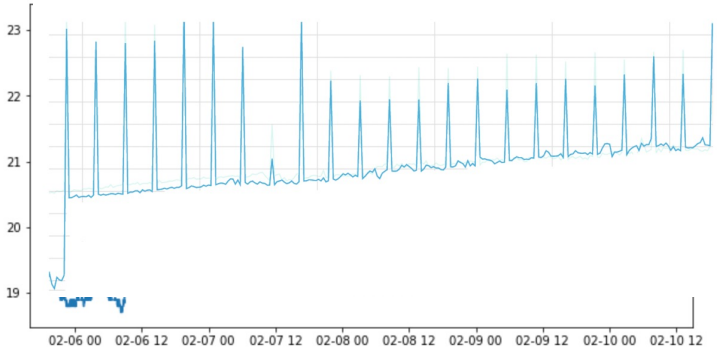
- Leverage service mesh / tools to get service interactions and the graph

Network Source	Network Destination	Average Count
Container A	Container B	35
Container C	Container B	140
Container B	Container D	3
…	…	…

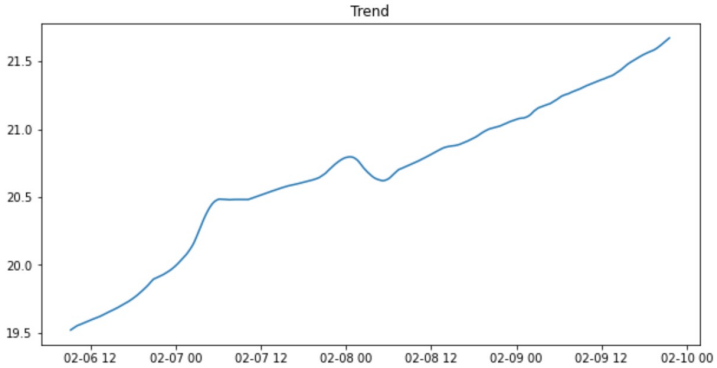




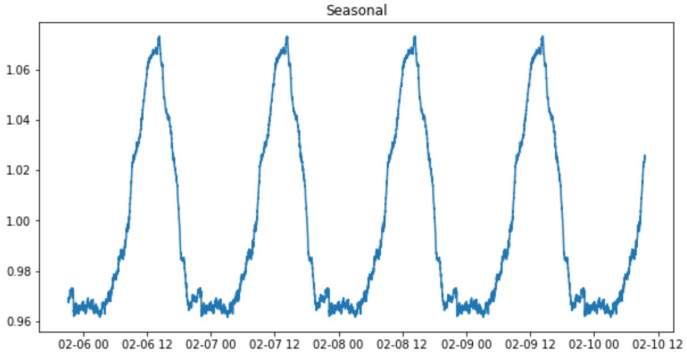
S-ARIMA


23
22
21
20
19
02-06 00
02-06 12
02-07 00
02-07 12
02-08 00
02-08 12
02-09 00
02-09 12
02-10 00
02-10 12

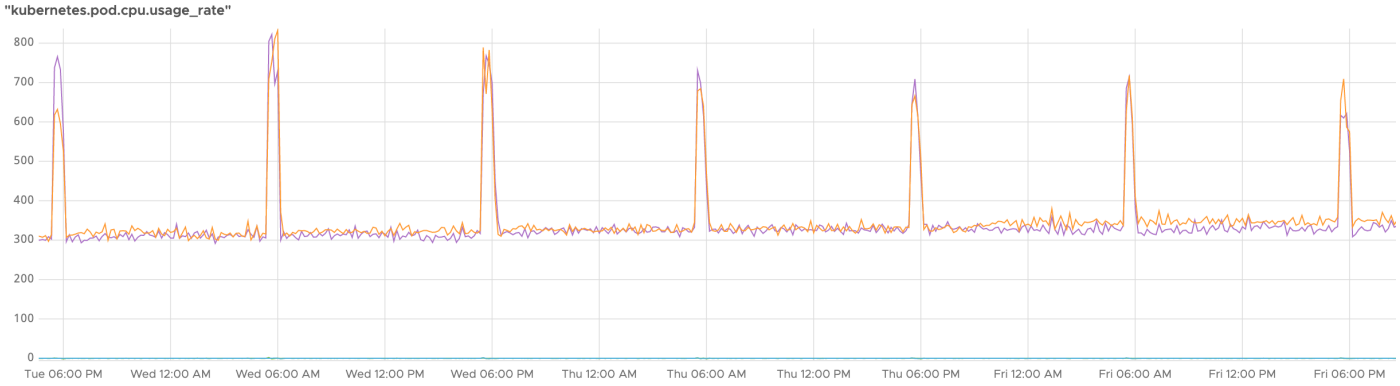
=

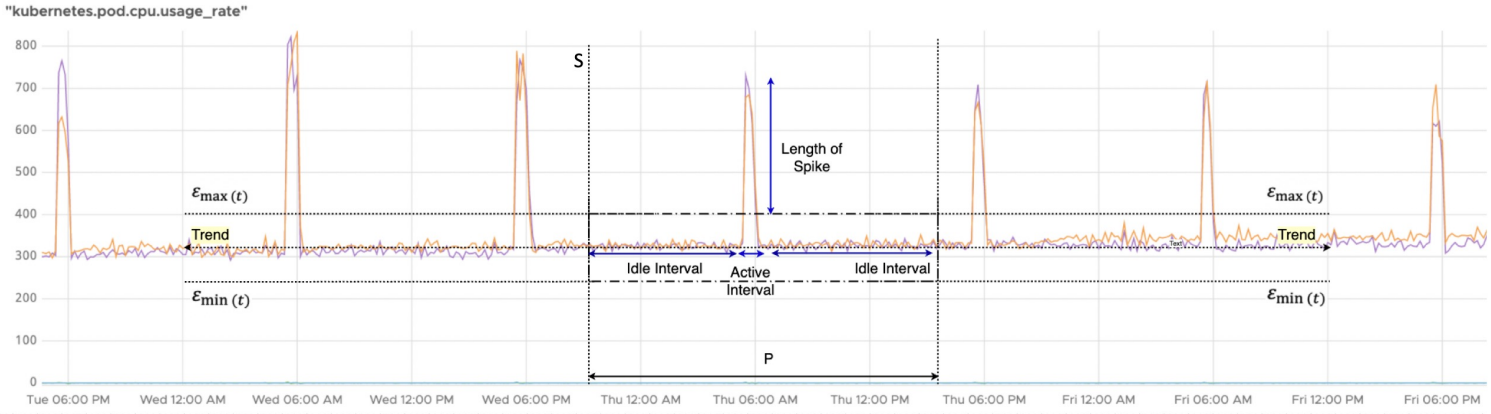

Trend
21.5
21.0
20.5
20.0
19.5
02-06 12
02-07 00
02-07 12
02-08 00
02-08 12
02-09 00
02-09 12
02-10 00

+

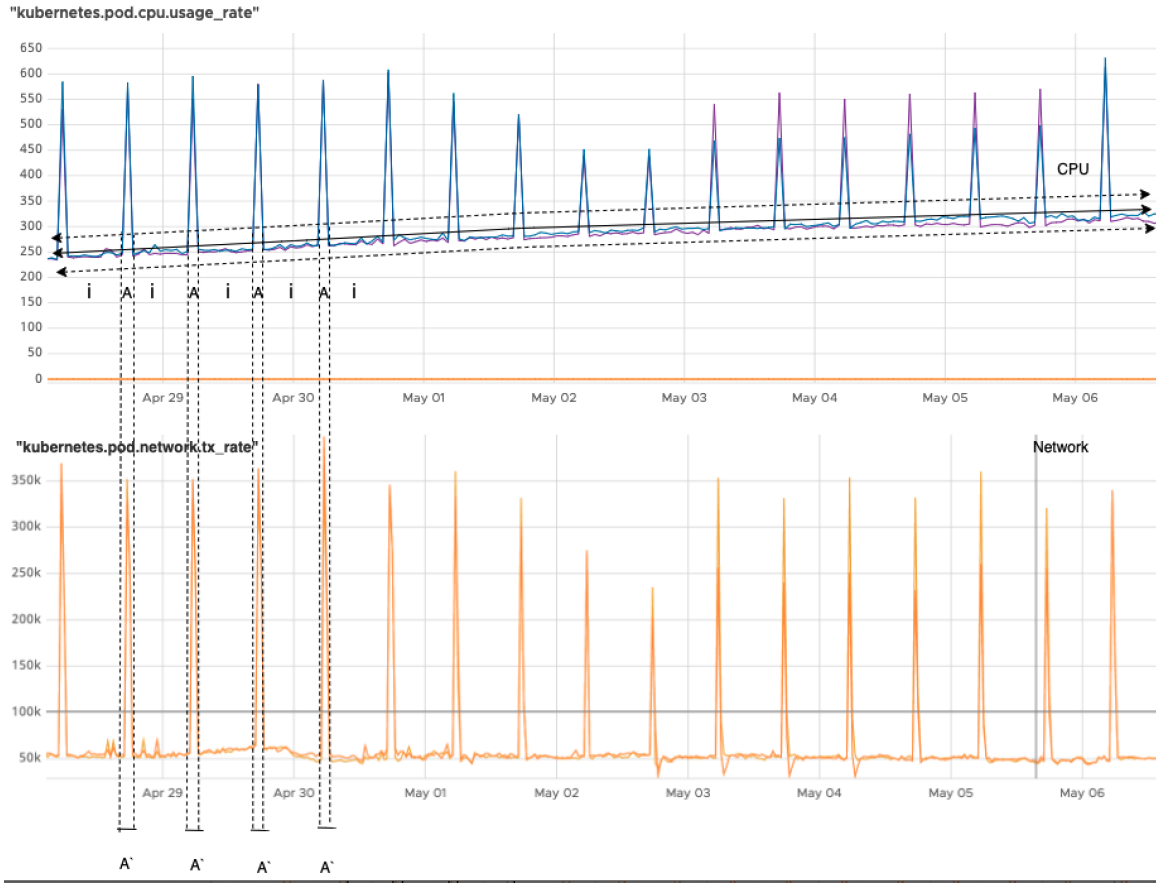

Seasonal
1.06
1.04
1.02
1.00
0.98
0.96
02-06 00
02-06 12
02-07 00
02-07 12
02-08 00
02-08 12
02-09 00
02-09 12
02-10 00
02-10 12

The evidence – resource usage



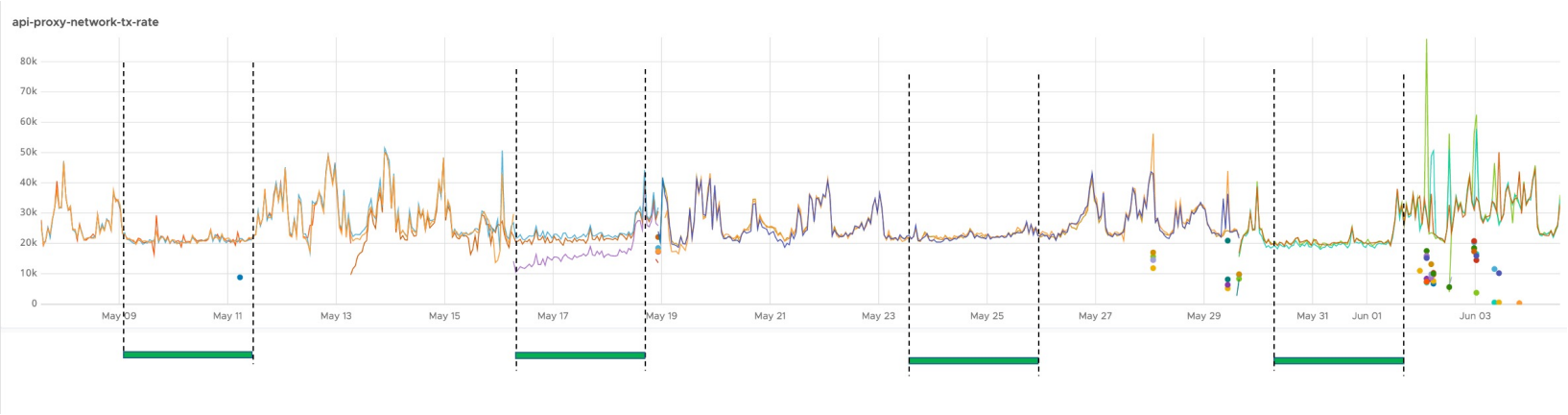


Identify & verify areas of active usage



- Identify overall period of spikes – when application is active
- Verify if all resources show the same picture

Identify options for low usage intervals



Application can be scaled down for resources OR moved to serverless to skip these intervals

The overall scheme

1. Seasonality Detection - Seasonal-ARIMA modelling
2. Spike Detection modelling and and filtering
3. Classification of active and idle intervals
4. Aggregation across resources and overlap detection

Helpers – AI and LLMs

- We leveraged LLM to generate the code for all the stages.

```
prompt = "Please complete the following steps:

    1. Data Preparation:
        a. Load the SQL schema provided in file <PLACEHOLDER> into a Pandas DataFrame for analysis.
        b. Corresponding data is provided in csv file in location <PLACEHOLDER>, sharded daily in the filename and filepath.
        c. Handle missing values and outliers as necessary.
        d. Check for stationarity and perform differencing if required.
    2. Model Selection:
        Use statistical methods (e.g., ACF/ PACF plots) to identify potential ARIMA parameters.
        Consider automatic model selection techniques if applicable.
    3. Model Fitting:
        Fit the selected S-ARIMA model to the data.
    4. Model Evaluation:
        Test ad tune the model across different data shards privided in the file.
        Evaluate the model's performance using metrics like RMSE, MAE, and MAPE.
    5. Forecasting:
        Generate forecasts for a specified future period and verify.
    6. Visualization:
        Visualize the original data, fitted model, and forecasted values."
```

Real world Testing

- Our environment -
  - 50+ services in the org,
  - 10 standard metrics considered for each service.

- Upon verifying the seasonality and modelling:
  - 19 real candidates with seasonal and bursty usages
  - 5 of which were the source of bursts and spiked usage in others

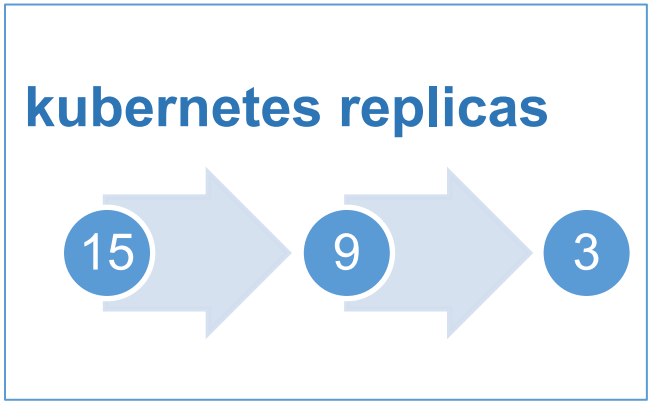
- Only 5 /50 services were evaluated for migrating towards serverless.

Remediation options

- Rearchitect option - split application into:
  - "Control plane" aka user serving micro-servers
  - "Batch processing" based serverless performing

- Reuse option - deploy same server based application as k8s job
  - Reduce instances of microservices
  - Introduce jobs with during previous intervals of high usage

An example:



- Per customer
- Handle periodic processing (explicitly invoked)

- For ad-hoc & regular API calls, etc.
- Eventually scaled down to 3

Results

- Saved close to **3200$** per month on infra costs just by refactoring these 3 applications
- **15% lesser** container failure rates and restarts across all the 19 application replicas.
- Able to detect a very resource intensive liveness probe in one application, which was fixed.
- Inherent ability to detect changes in patterns and detect anomalies - like changes in expected patterns.

November 12, 2024
Salt Lake City

Yash Bhatnagar
Software Engineer
Google