SCHIRESON

Predicting Scale with Prometheus and ML

Chris Dutra, Schireson

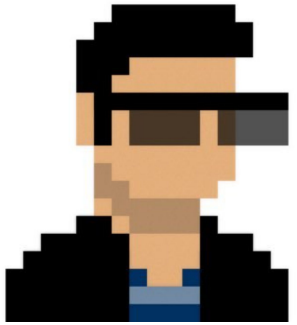
Chris Dutra

Director, Site Reliability Engineering

chris@schireson.com

@dutronlabs

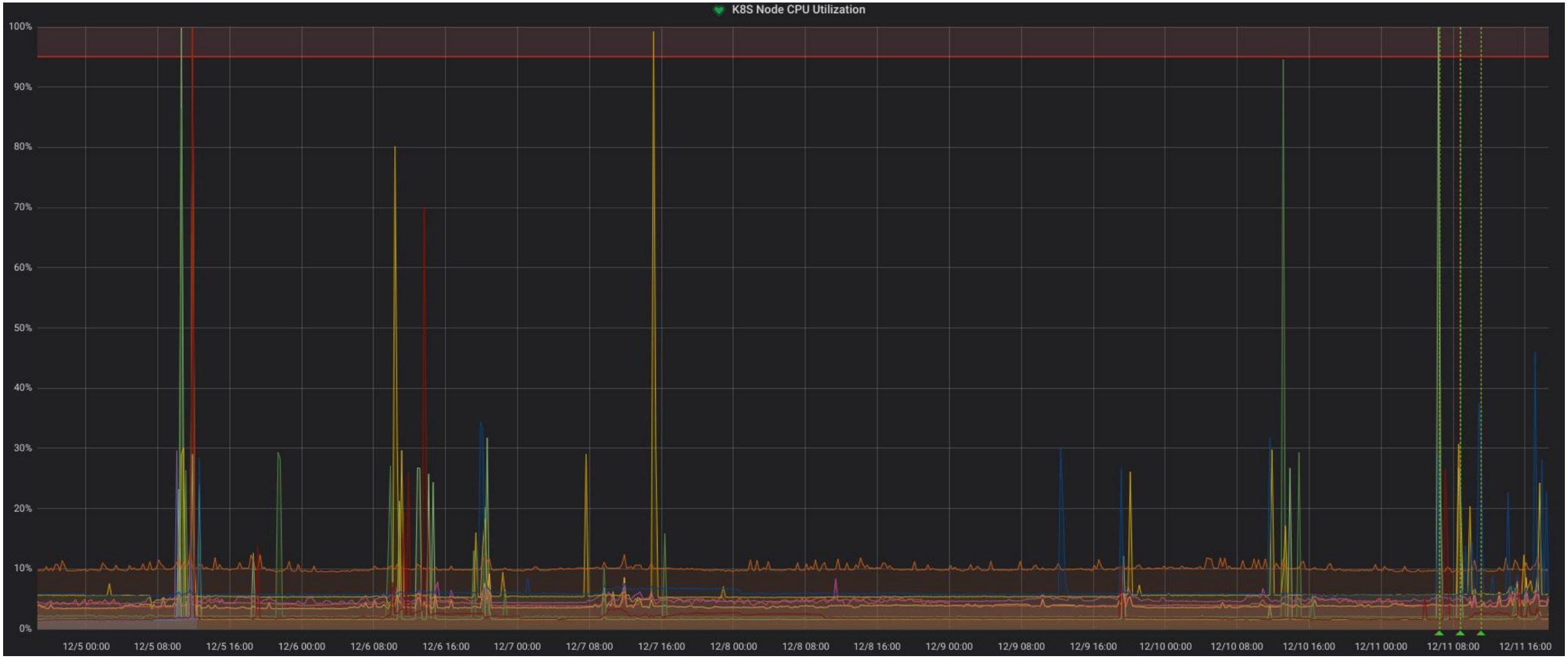
@chrisdutra

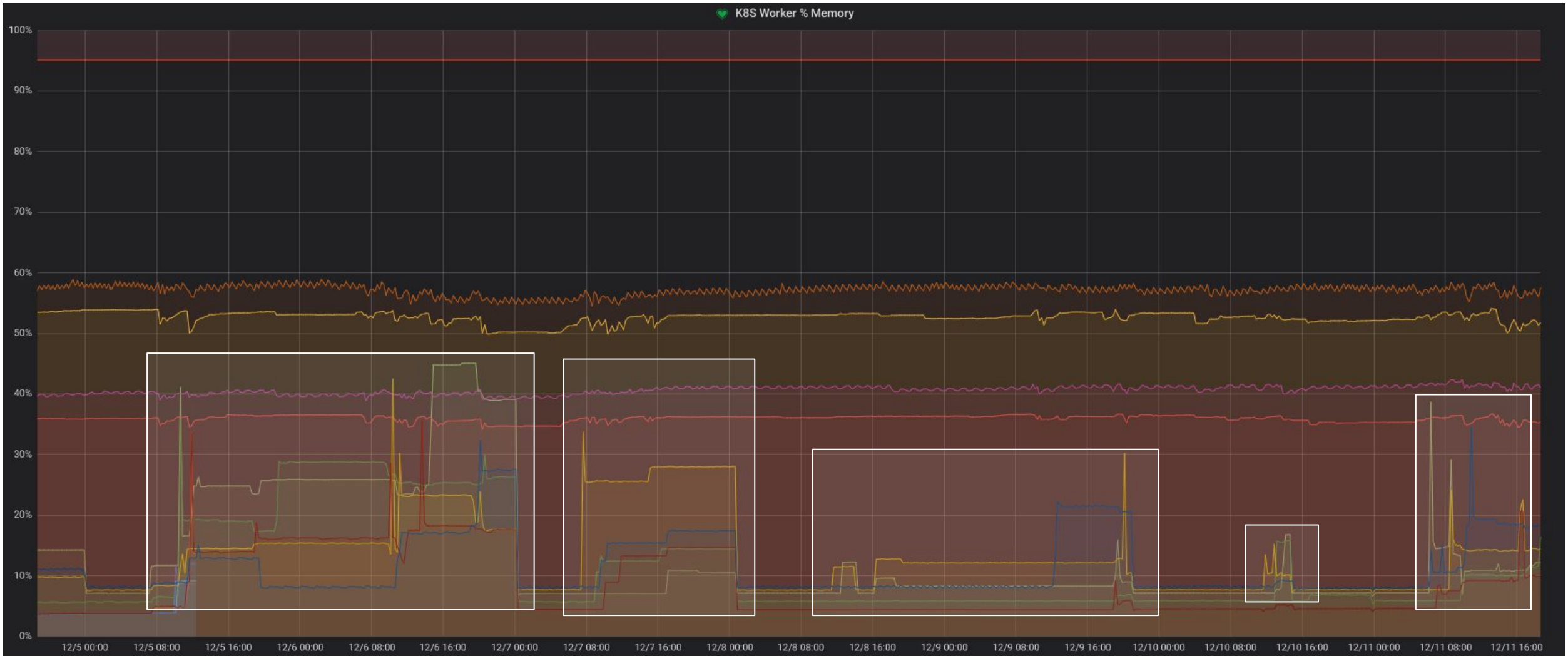



SRE @SCHIRESON

Moving Data Science Processes to Production

- Code migration from “Jupyter to Container”
- Achieve similar performance as laptop-driven simulations
- Provide “data science as a service” to clients


K8S Node CPU Utilization
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
12/5 00:00
12/5 08:00
12/5 16:00
12/6 00:00
12/6 08:00
12/6 16:00
12/7 00:00
12/7 08:00
12/7 16:00
12/8 00:00
12/8 08:00
12/8 16:00
12/9 00:00
12/9 08:00
12/9 16:00
12/10 00:00
12/10 08:00
12/10 16:00
12/11 00:00
12/11 08:00
12/11 16:00


K8S Worker % Memory
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
12/5 00:00
12/5 08:00
12/5 16:00
12/6 00:00
12/6 08:00
12/6 16:00
12/7 00:00
12/7 08:00
12/7 16:00
12/8 00:00
12/8 08:00
12/8 16:00
12/9 00:00
12/9 08:00
12/9 16:00
12/10 00:00
12/10 08:00
12/10 16:00
12/11 00:00
12/11 08:00
12/11 16:00

Moving Data Science Processes to Production

- Code migration from “Jupyter to Container”
- Achieve similar performance as laptop-driven simulations
- Provide “data science as a service” to clients

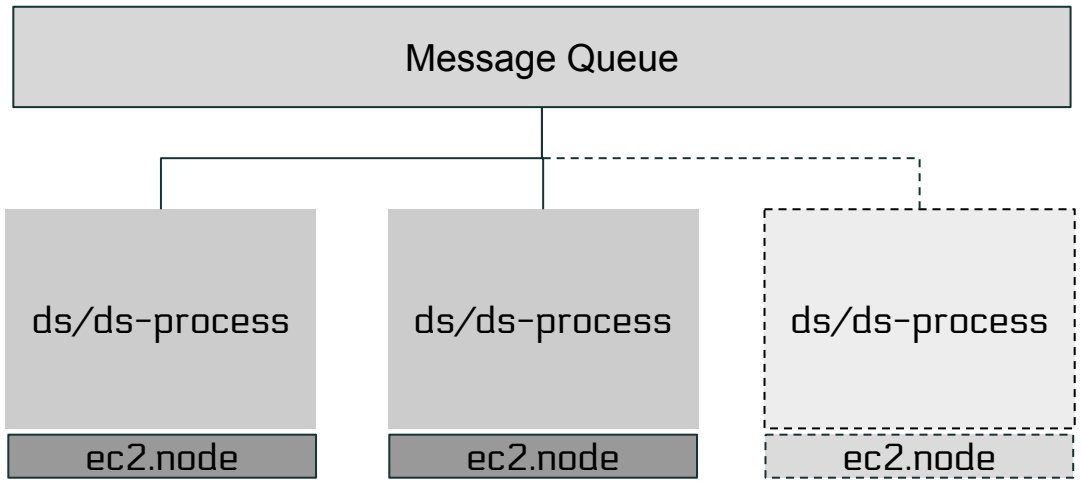
But…

- CPU/Memory utilization is non-deterministic!
- DS Processes are long running and resource hungry!
- Can we follow same SDLC as other engineering work?
- How does this *work* in a live environment?

Moving Data Science to Production

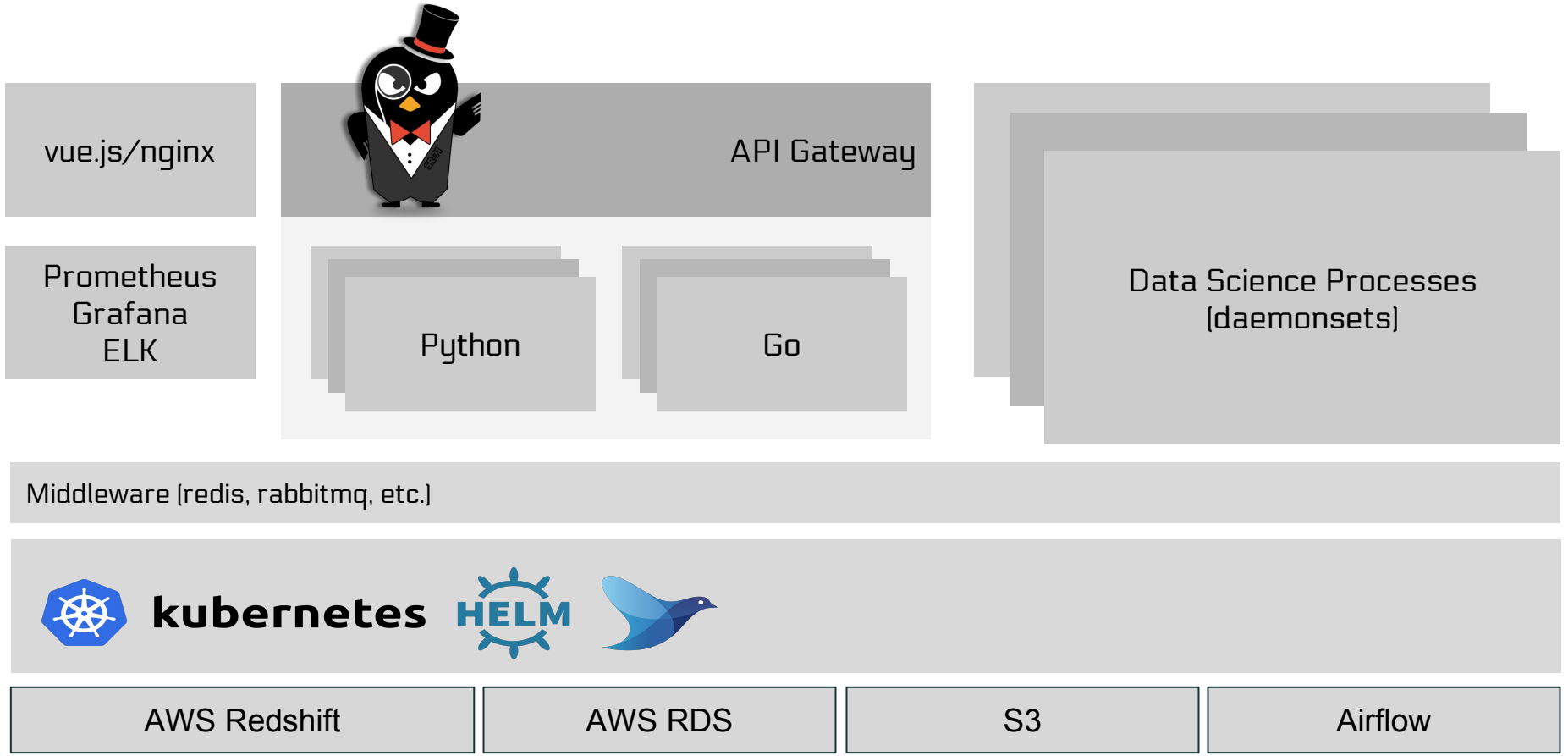
Solution: **daemonsets**!

- DS process gets node resources with light scheduling overhead
- Isolate resources via nodeSelector, node labels
- Scaling currency is now infrastructure based
- Retain orchestration capabilities!



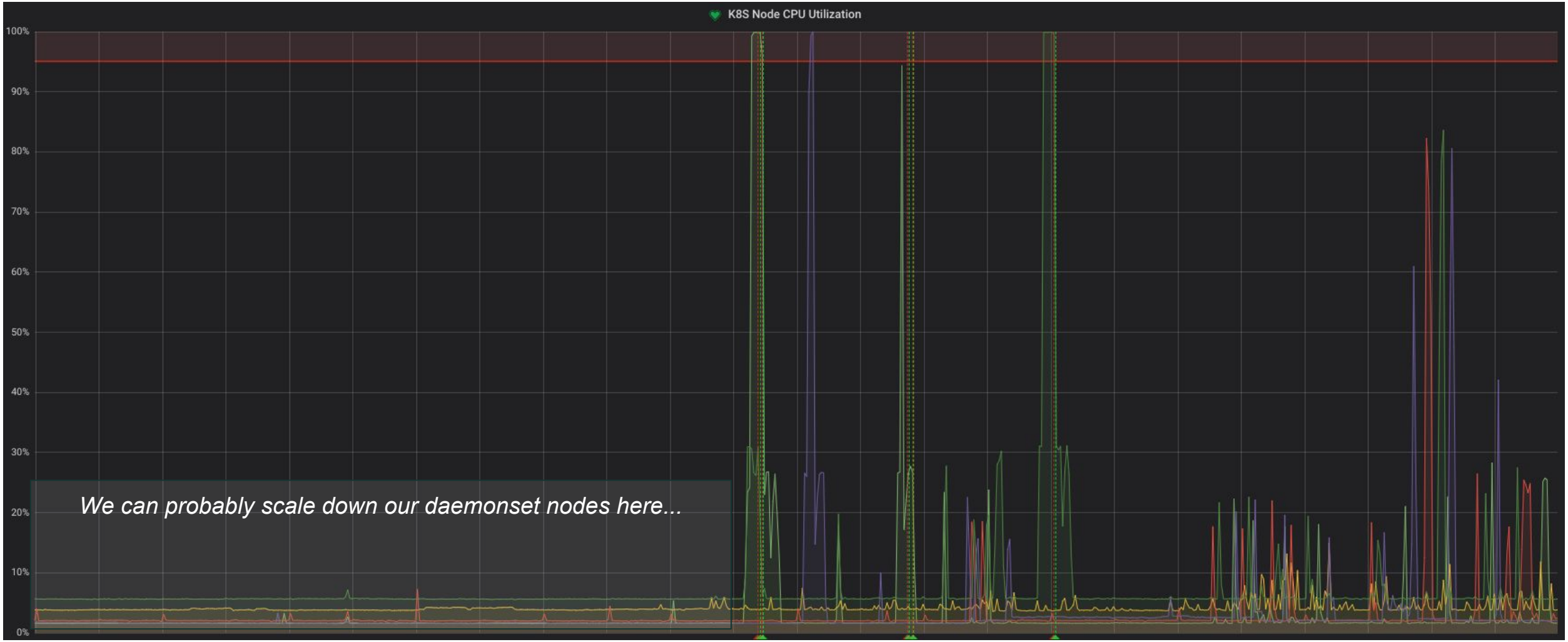
```
chris@chris-Oryx-Pro:~$ kubectl get nodes -L runtime
NAME                           STATUS    ROLES     AGE       VERSION     RUNTIME
ip-172-33-50-63.ec2.internal   Ready     node      1d        v1.10.11    ds-process
ip-172-33-51-111.ec2.internal  Ready     node      1d        v1.10.11    api
ip-172-33-63-121.ec2.internal  Ready     master    1d        v1.10.11
ip-172-33-64-86.ec2.internal   Ready     node      1d        v1.10.11    api
ip-172-33-70-185.ec2.internal  Ready     node      1d        v1.10.11    api
ip-172-33-75-9.ec2.internal    Ready     node      1d        v1.10.11    ds-process
ip-172-33-76-86.ec2.internal   Ready     node      1d        v1.10.11    ds-process
ip-172-33-86-160.ec2.internal  Ready     node      1d        v1.10.11    api
ip-172-33-87-96.ec2.internal   Ready     node      1d        v1.10.11    ds-process
ip-172-33-92-57.ec2.internal   Ready     node      1d        v1.10.11    ds-process


chris@chris-Oryx-Pro:~$ kubectl get ds -n kube-public
NAME         DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR        AGE
ds-process   5         5         5         5            5           runtime=ds-process   1d


chris@chris-Oryx-Pro:~$ kubectl get pods -n kube-public -l k8s-app=ds-process -o wide
NAME               READY     STATUS    RESTARTS   AGE       IP               NODE                           NOMINATED NODE
ds-process-4n9sc   1/1       Running   0          19h       100.112.117.224  ip-172-33-50-63.ec2.internal   <none>
ds-process-jnjtx   1/1       Running   0          19h       100.101.80.71    ip-172-33-75-9.ec2.internal    <none>
ds-process-jt88k   1/1       Running   0          19h       100.97.212.102   ip-172-33-76-86.ec2.internal   <none>
ds-process-kc5kp   1/1       Running   0          19h       100.98.122.28    ip-172-33-92-57.ec2.internal   <none>
ds-process-rk8dj   1/1       Running   0          19h       100.97.127.136   ip-172-33-87-96.ec2.internal   <none>
```

Schireson Today

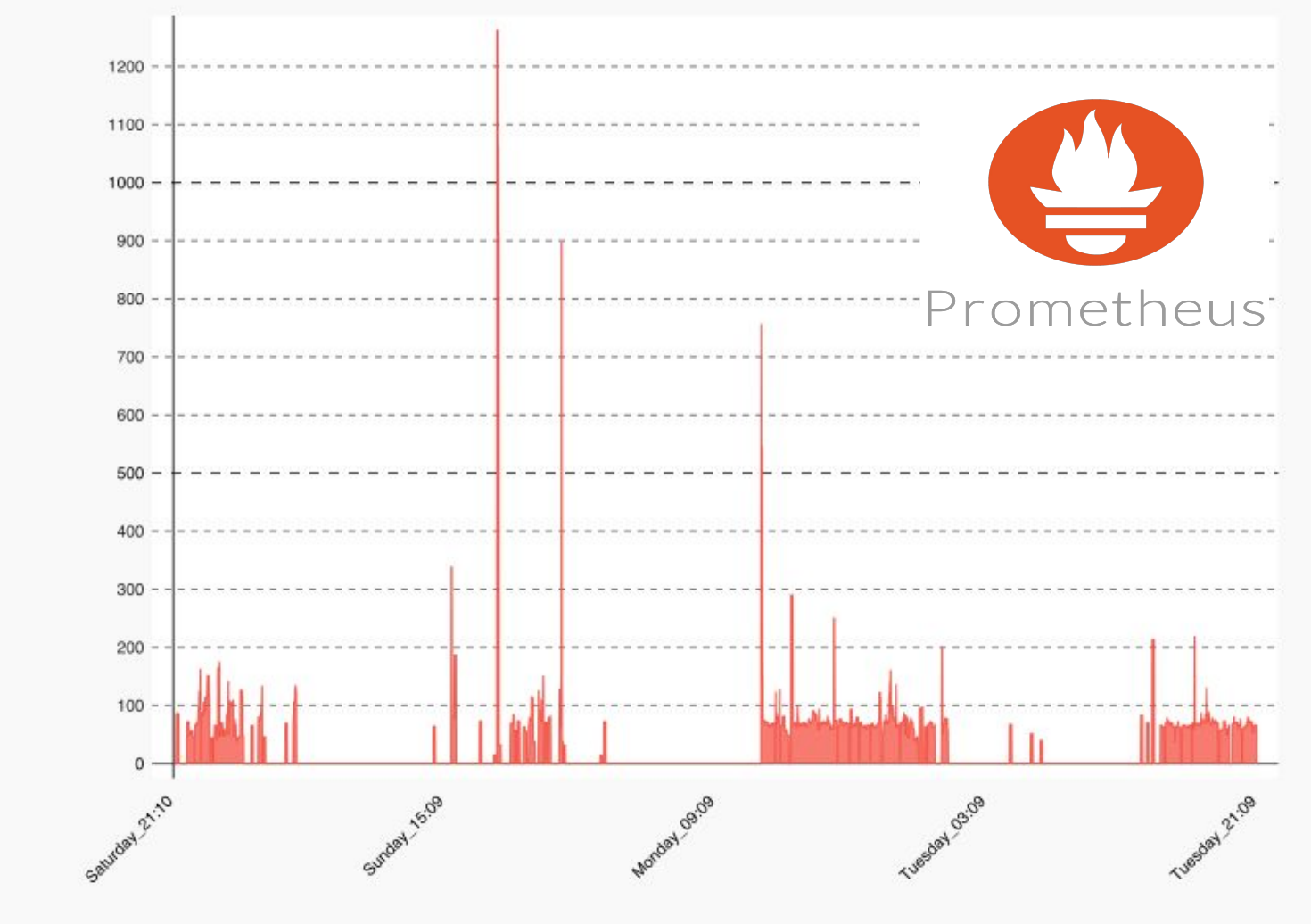


How do we scale?

… while minimizing SRE load?	(important to me)
… while keeping cloud spend in check?	(important to boss)
… AND while providing a great experience?	(important to client)


K8S Node CPU Utilization
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
We can probably scale down our daemonset nodes here...

Solution: Time Series Forecasting



Anticipating changes to key metrics can help an SRE team:

- Right size infrastructure
- Provide more redundancy during peak times
- Reduce cost during non-busy periods

There are *many* ways to forecast, but we chose **LSTM**.

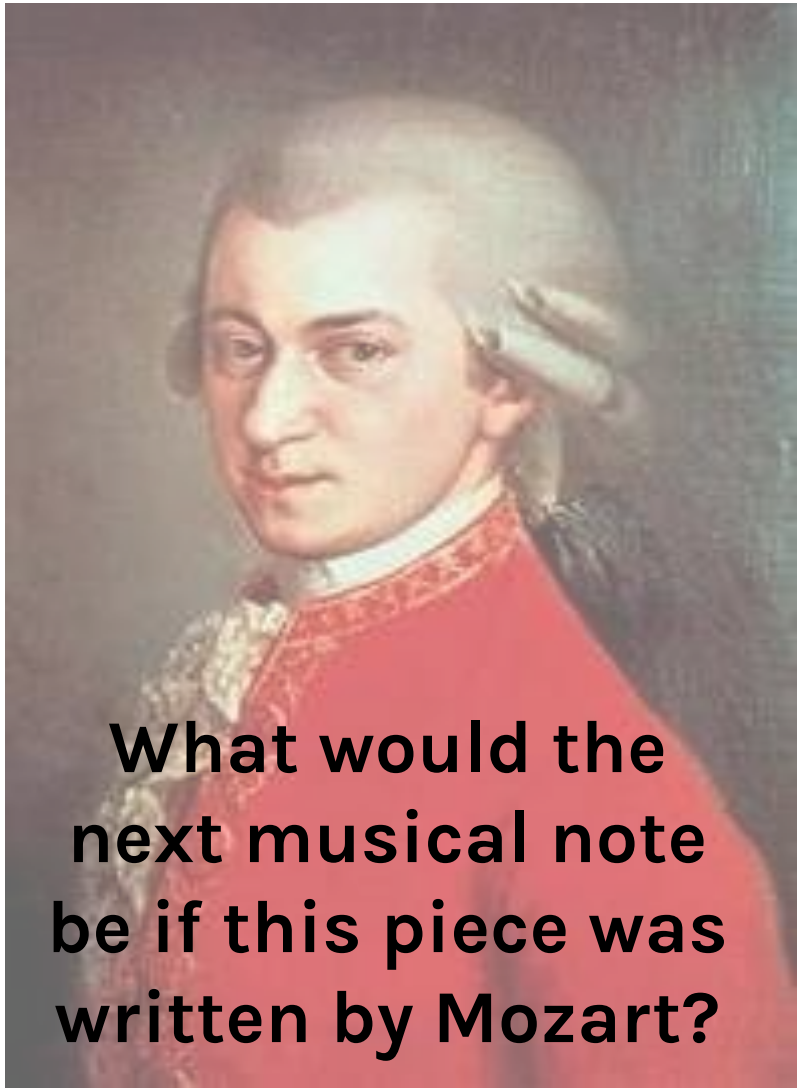
LSTM Model

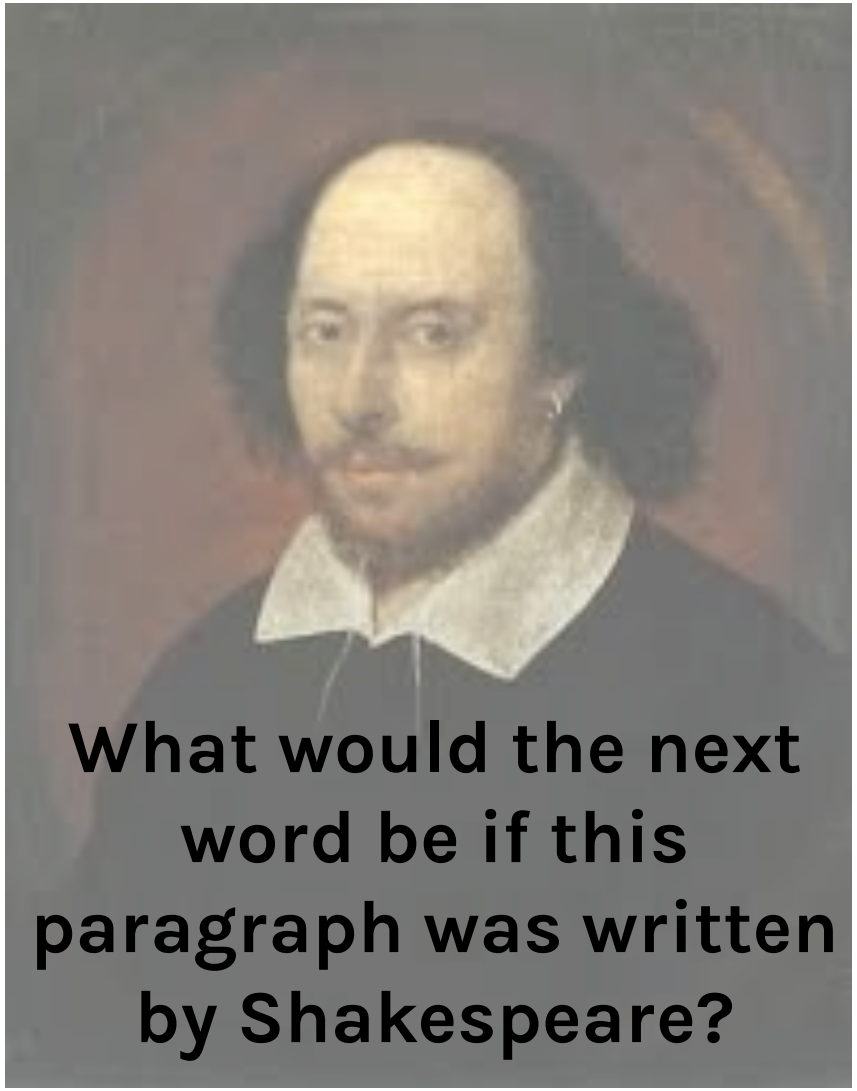
- Stands for **Long Short-Term Memory**
- Recurrent Neural Network (RNN) designed to recognize patterns in **sequences of data**

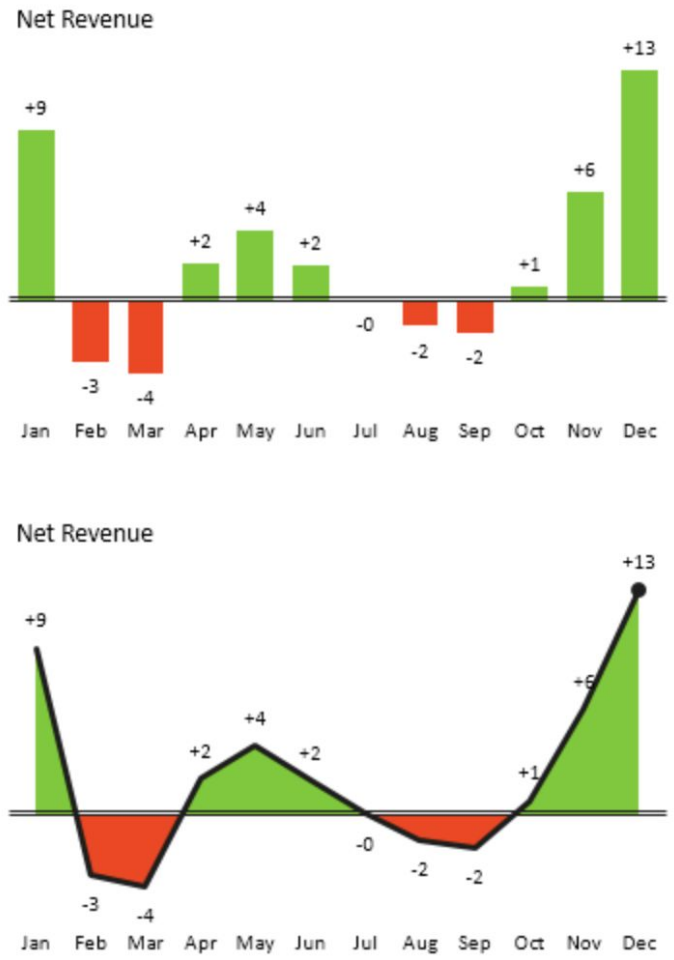
Can you predict the next word in this paragraph?

Hi, my name is Chris Dutra, and thank you for joining me at this conference for software engineers and Kubernetes experts. Today, I would like to talk to you about containers!

Additional LSTM Examples

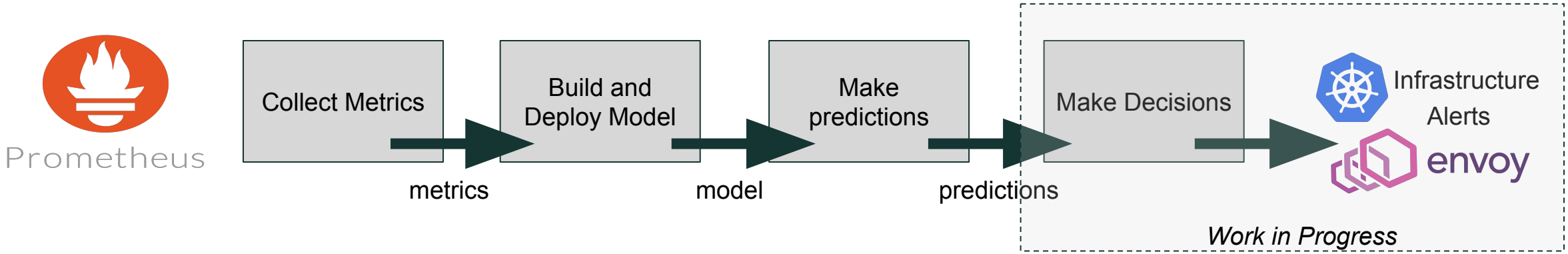






Time Series Forecasting

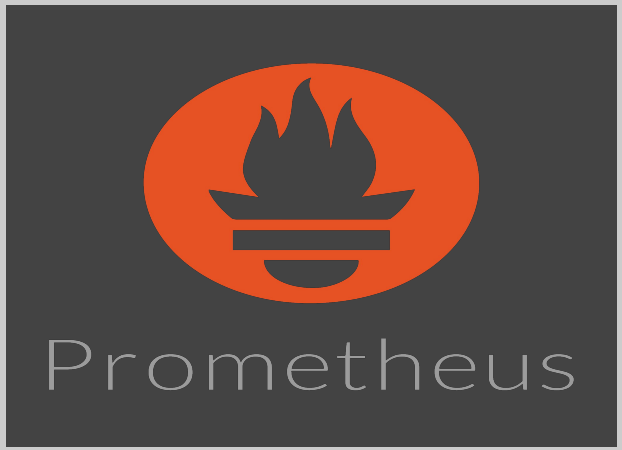
End to End Flow



Train and Deploy the LSTM Model

ACQUIRE

- Collect data range from Prometheus
  - Large enough, **but not too big**
- Minor formatting of data into dataset
- Example metrics:
  - Replica Counts
  - Requests per Second
  - Upstream Latency (Envoy)
  - Infrastructure
  - ...

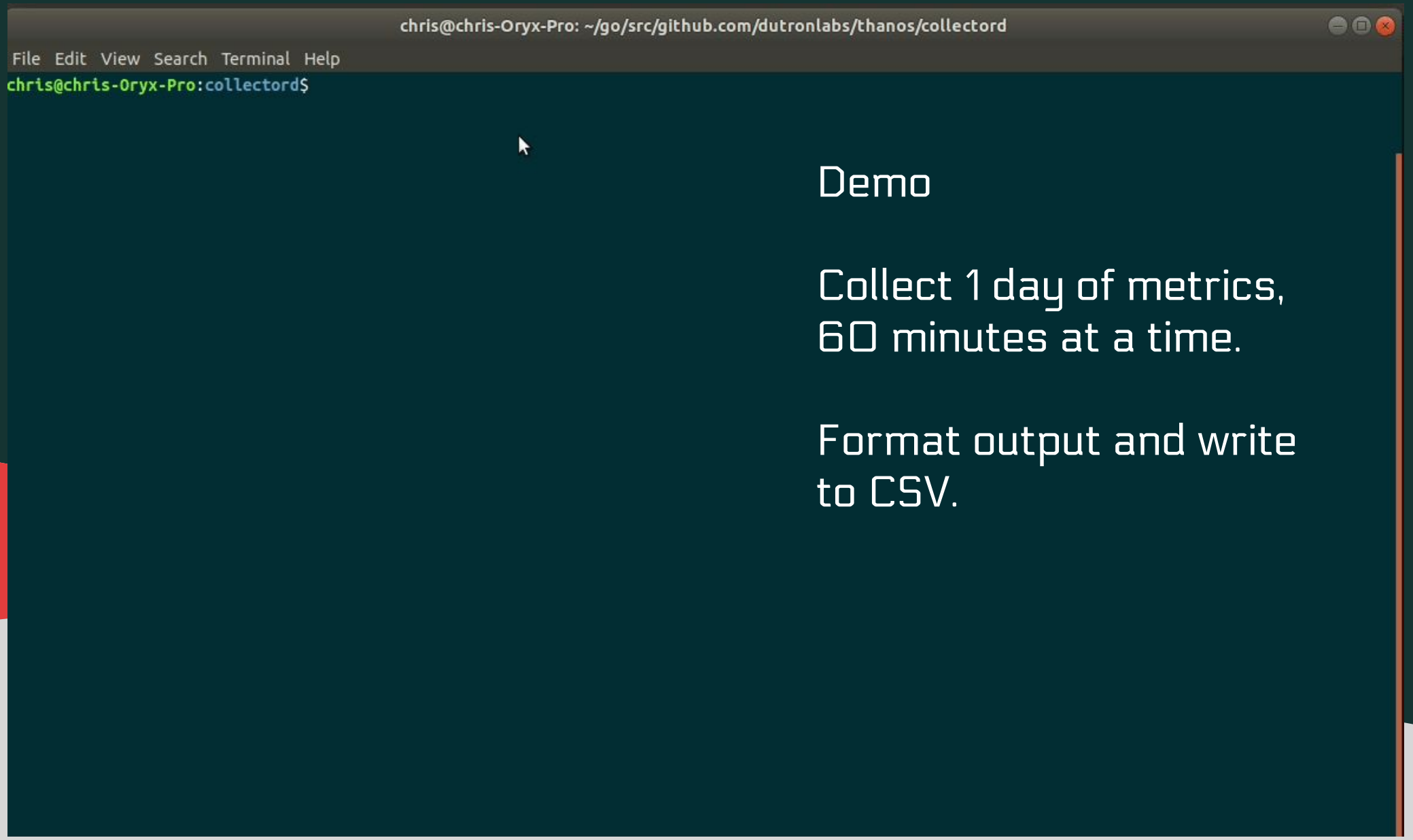


Collecting Metrics in Prometheus

- Batch your queries when pulling data from Prometheus.
- Otherwise, you might run into errors like:

  Error executing query: exceeded maximum resolution of 11,000 points per timeseries. Try decreasing the query resolution (?step=XX)

```go
func getDataSet(c Config) {
    prometheusClient := &http.Client{
        Timeout: 10 * time.Second,
    }
    // the ending timestamp is right now
    timeNow := time.Now().Unix()
    // get the starting timestamp (look back c.NumDays)
    startTime := time.Now().AddDate(0,0,-1*c.NumDays).Unix()
    // create slice of model outputs, which maps to Prometheus JSON output.
    batchedOutputs := []models.POutput{}
    // the BatchInterval value is in minutes, convert to seconds to work with Unix time.
    minuteInterval := c.BatchInterval*60

    for i := startTime; i < timeNow; i=i+(minuteInterval) {
        fmt.Printf("Fetching records from unix time %d to %d \n", i, i+minuteInterval)
        url := buildURL(c,i,i+minuteInterval)
        response, err := prometheusClient.Get(url)
        if err != nil {
            panic(err.Error())
        }
        defer response.Body.Close()
        decodedOutput := models.POutput{}
        if decodedOutput.Error != "" {
            fmt.Printf("Error in response from Prometheus: %v", decodedOutput.Error)
            // fast fail here. this most likely means our query went over 11K timeseries
            panic(err.Error())
        }
        json.NewDecoder(response.Body).Decode(&decodedOutput)
        batchedOutputs = append(batchedOutputs, decodedOutput)
    }
    util.WriteModelSliceToCsv(batchedOutputs, c.CSVFile)
    fmt.Println("Collection complete!")
}
```



Demo

Collect 1 day of metrics, 60 minutes at a time.

Format output and write to CSV.

Train and Deploy the LSTM Model

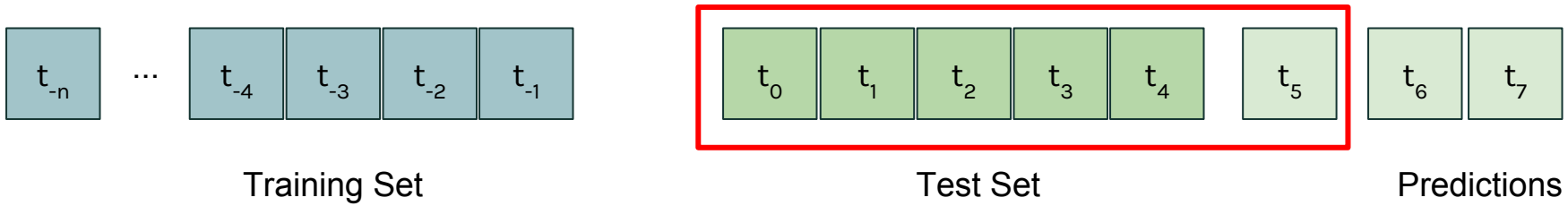
TRAIN & TEST

- Split data into test and training sets
- Transform data (supervised data, scaling)
- Fit model against training set
- Evaluate model against test set, benchmarks
- Deploy model


```
fitted_lstm_model = lstm_model.fit_lstm(train_scaled,
                                        batch_size=1,
                                        nb_epoch=int(config["lstm_parameters"]["epochs"]),
                                        neurons=int(config["lstm_parameters"]["neurons"]))

fitted_lstm_model.save(os.path.join(pickle_model_dir, "lstm_model.h5"))
```

Generate Predictions

- Choose a smaller dataset that represents recent history of your metric
  - for example, 1 hour of data to predict the next 10 minutes
- Split dataset into training/test datasets
- Forecast data on training set to gather state
- Generate predictions, using test set



- **Important** - the further out predictions are generated, the higher rate of error must be accounted for.
- Generally speaking, predictions 15 minutes into the future have a *lower* margin of error than 60 minutes.

Actionable Data!

What can we conclude about this data?

- *Anticipating little to no traffic*

What's actionable about this data?

- *Scale down replicas*
- *Scale down infrastructure*
- *…*

Predicted RPS

time	prediction	actual
t0 (present)	0.0	0.0
t1 (+1min)	0.0	
t2 (+2min)	0.0	
t3	0.0	
t4	0.0	
t5	0.0	
t6	0.0	
…		
t15 (+15min)	0.0	

Actionable Data!

What can we conclude about this data?

- *Lots more traffic incoming!*

What's actionable about this data?

- *Scale up replicas*
- *Scale up infrastructure*
- *…*

Predicted RPS

time	prediction	actual
t0 (present)	12.5	11.7
t1 (+1min)	10.0	
t2 (+2min)	85	
t3	150	
t4	170	
t5	215	
t6	250	
…		
t15 (+15min)	5000	

Making Decisions

Example: Stateless Resources (Deployment)

- Define thresholds for your resources
  - RPS below 50, above 2500
- Define minimum and maximum gates for your resources
  - No less than 3, no more than 15 replicas
- Determine how long to wait (sleep) before analyzing the next set of predictions.

```
// get predictions for the next x minutes loaded into an array
for {
    predictions, err := getPredictions(p)
    if err != nil {
        fmt.Printf("Unable to load predictions")
    }
    // if the prediction is above our scaleupthreshold,
    // then let's scale up replicas
    if predictions[p.Horizon-1] > p.ScaleUpThreshold {
        ScaleReplicas(p, 1)
    }
    // conversely, if prediction is below our scaledownthreshold,
    // let's scale down replicas
    if predictions[p.Horizon-1] < p.ScaleDownThreshold {
        ScaleReplicas(p, -1)
    }
    // then we sleep until the next interval
    time.Sleep(time.Duration(p.SleepInterval) * time.Minute)
}
```

Future Work

- *Can we do better*?
  - Tuning
  - Multivariate LSTM
- Explore Reinforcement Learning (q-learning)
- Forecast other key areas
  - Saturation
  - Cluster Management

How low-touch can we get our kubernetes clusters?

Future Work

CRD - **P**redictive **A**uto-**S**caler

- Similar to HPA, but with the workflow outlined above
- Ability to interact with k8s resources, infrastructure

```
kubectl create pas ...
```

Acknowledgements

- Eben Esterhuizen and Schireson Data Science Team!
  - Kubernetes, Prometheus, & Envoy Communities

THANK YOU