Metrics

Proof of Technology - Monitoring

Christian Boin

March 21, 2022

Goal: Make Observability Easy

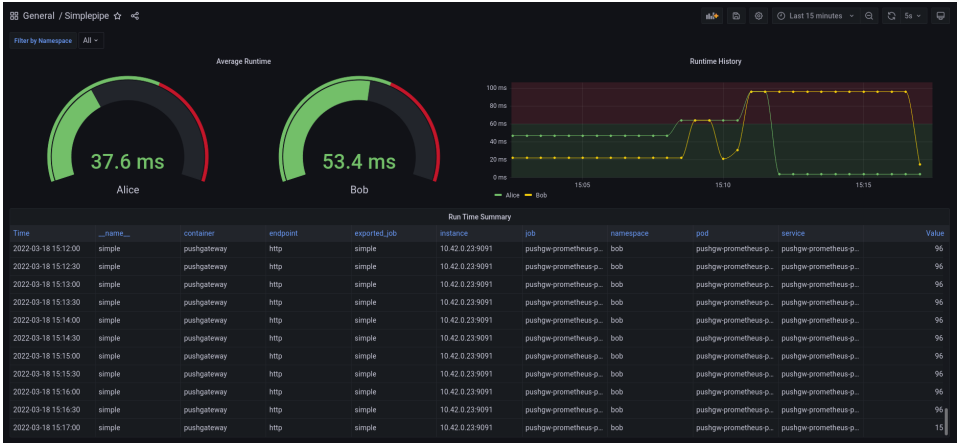

Figure: Administrator dashboard example monitoring application run-times.

Purpose

- To develop a proof of technology illustrating:
  - How a metrics gathering application can be designed and implemented using Prometheus and Grafana
  - How metrics can be generated within data science projects for use with Prometheus
  - Standardized monitoring across Argo Workflows and Seldon

Project Technologies

- Argo Workflows
- Prometheus Pushgateway
- Prometheus
- Grafana
- Python
- Docker
- Kustomize
- Helm
- Kubernetes

TL;DR

- Prometheus is a pull based monitoring system, however Prometheus provides a Pushgateway which allows pushing of metrics via a REST-like interface or an SDK
- Installing a Prometheus Pushgateway per user namespace can allow each user to push metrics since Service Monitors can allow Prometheus to scrape each Pushgateway
- An application, Simplepipe, was designed as to make use of the Pushgateway Python SDK in a standardized simple to use format.

Design

- Utilizing the Prometheus Pushgateway allows two monitoring models to come to mind: Concurrent (real time) or Sequential

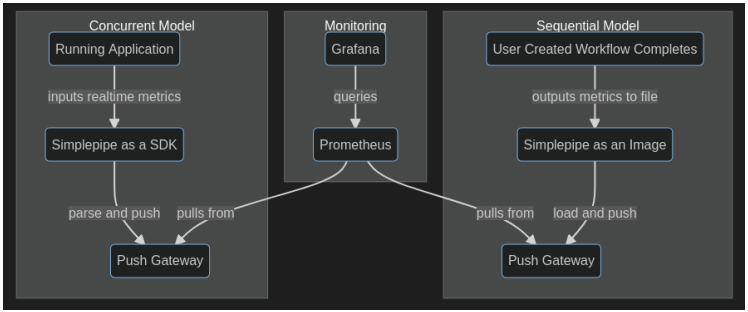

Figure: High level flow diagram of sequential versus concurrent models

Usage: Concurrent Design as a Python package

- A concurrent model would likely require users to manage a Prometheus Pushgateway within their application
- To standardize metric creation, users could import Simplepipe as an SDK and push metrics:

```
metrics = [{"name": "example_metric",
            "description": "metric for example purposes",
            "value": 0,
            "metric_type": "gauge"}]
collector = MetricCollector("http://localhost:9091")
collector.parse_and_push(metrics=metrics, job_name="example_from_dict")
```

Usage: Sequential Design

Currently, a sequential design is implemented:

- Useful for monitoring information across workflows such as:
  - run time
  - data processing information (number of images, data volume processed)
  - accuracy of ML models
- Users would output metrics in a predefined format
- Simplepipe would be responsible for loading and pushing metrics to the Pushgateway
- We implemented the sequential design using Argo workflows configured to use Minio as an artifact repository

Usage: Sequential Design (cont.)

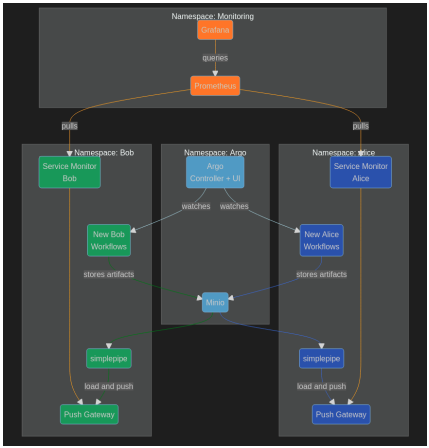

Figure: Detailed design of sequential model

Usage: Sequential Design (cont.)

- Users (Alice and Bob) output artifacts within workflows:
- An artifact can be imported to Simplepipe within a Workflow CRD:

```
inputs:
  artifacts:
    - name: file
      path: "/tmp/metric-in.json"
      s3:
        key: metric-json
container:
  image: simplepipe:latest
  imagePullPolicy: IfNotPresent
  command: [python3]
  args: ["app.py", "/tmp/metric-in.json", "simple"]
```

Example Scenario: Sequential Design

- An administrator wants to gain insight on application run times for two users: Alice and Bob
- The administrator instructs Alice and Bob to use Prometheus, and provides them a docker image of Simplepipe, with instruction on how to format their metrics as inputs to Simplepipe
- Alice and Bob time their applications, and output the time elapsed into proper metric format for Simplepipe
- The administrator creates a dashboard in Grafana, querying Prometheus for Alice and Bob's metrics

Example Scenario (cont.)

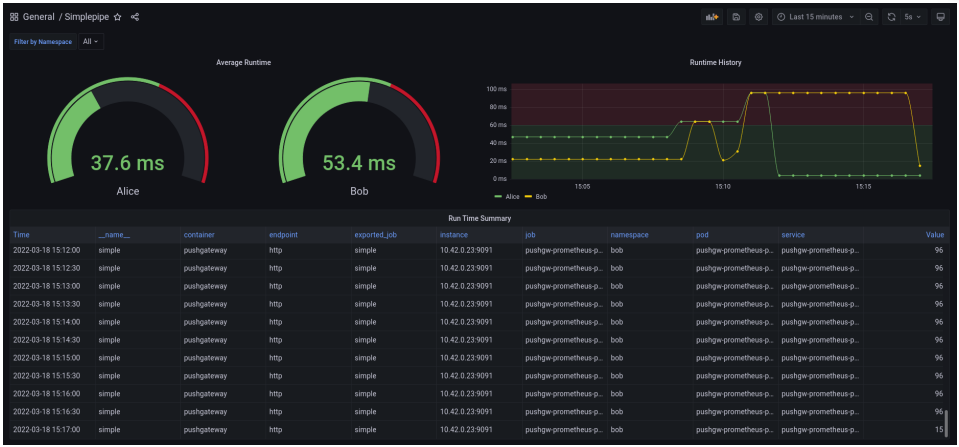

Figure: Administrator dashboard example monitoring application run-times.

Potential Applications

Sequential Design

- Any metric useful to be known once a workflow terminates:
  - Number of images processed in an image processing pipeline
  - Runtime of time critical workflows
  - Metrics related to training ML models such as accuracy
  - Memory usage

Concurrent Design

- Any metric needed to known in real time for long lived applications
  - DRL training often takes days, and knowledge of how the model is performing would be useful for determining if training should continue or can be abandoned

Potential Applications (cont.)

Sequential and/or Concurrent Designs

- Implementation within AAW would allow for users to add metrics to their applications with little to no effort
- Alerting for metrics to notify when something is failing:
  - Expected values can be set in Grafana and thus alerting is possible when values fall outside the threshold
- This work could also provide a standardized format for using metrics within Argo Workflows and Seldon

Future Work

- Multi-tenancy with Prometheus and Grafana is challenging, and solutions will need investigation
- Defining standard metrics for a consistency across multiple applications
- Standardizing dashboards (can be done with configMaps in k8s)

Questions?