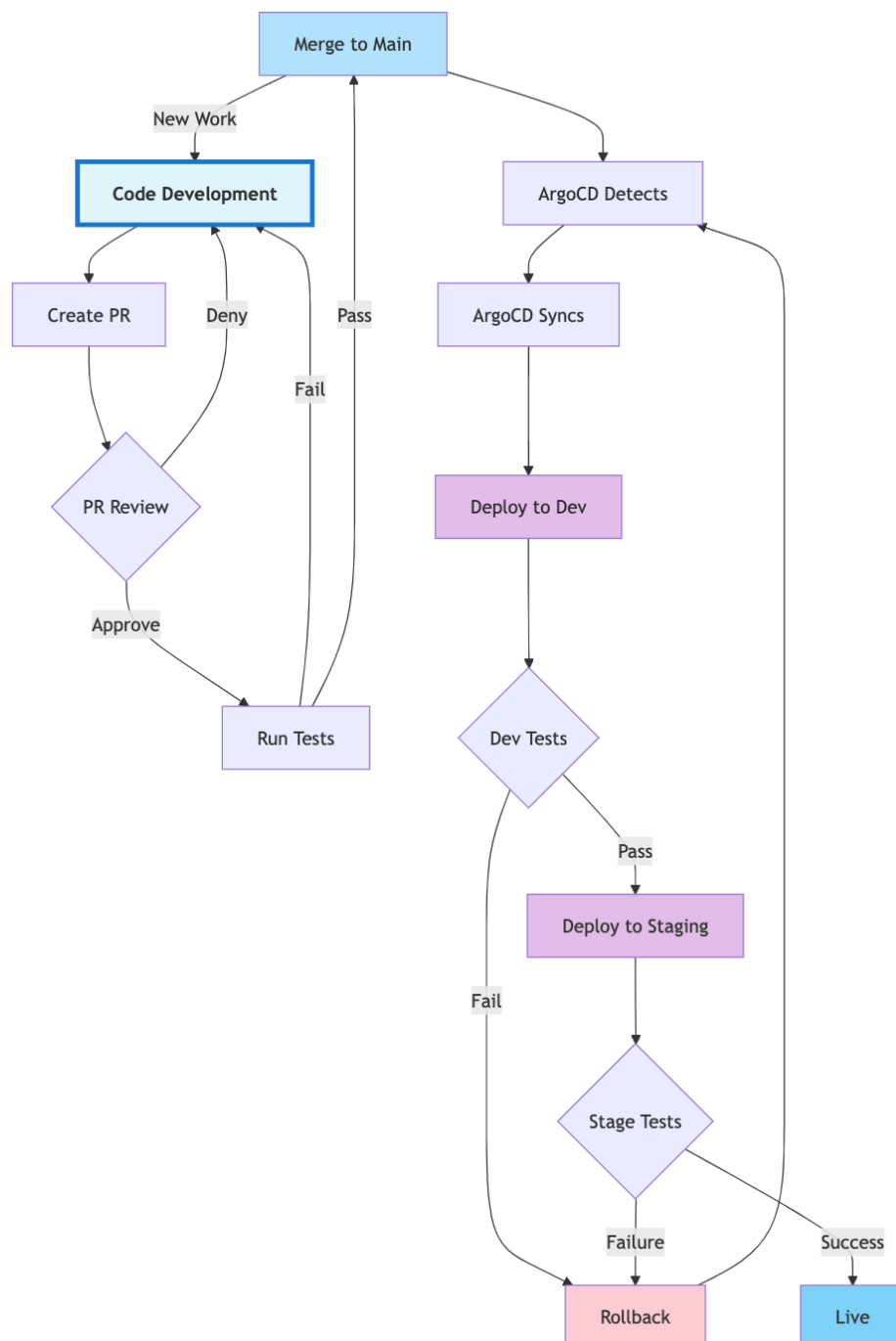


## Alertstack - Alert, Metric and Rule Creation and testing

This mini project creates a fully self contained Observability Stack locally on your machine. The main purpose is to provide an isolated environment to aid development and testing of rules, alerts and configs that make use of [Prometheus](#), [Alertmanager](#) and [Grafana](#).

Typically in different environments, like staging and production, the [Alert Rules](#), [Groups](#), [Routes](#) and [Receivers](#) are all tightly coupled in a handful of files. These rules/configs can be of different vintages and would cover numerous old and new services. It is not uncommon for there to be a wide range of styles apparent in the Alert/Rules definitions that have been developed over the years.

This test environment described here is a standin environment representation we can use to validate any major new work/definitions or modification to our existing Alerts/Rules/Receivers. So, remembering that these observability resources are implemented as yaml files and configs, they will be managed via GitOps and deployed to an environment to test properly, just like actual code.



By using this standalone `alertstack` environment we can add/modify the observability resources locally, and allow real testing of syntax and correctness *without* having to go thru many cycles of develop, merge, deploy, test, fix iterations to staging git first, as described above.

Below I detail four distinct componants that comprise this alertstack project. These components run via [docker-compose](#) using the relevent containers.

As well as [Prometheus](#), [Alertmanager](#) and [Grafana](#) you will see something called [pingpong](#) .

## pingpong

This is a service/application written in golang. It provides test [integration endpoints](#) for /pagerduty /slack and /webhooks. It also provides a proxy stub for the [/metrics](#) API endpoint so we can see what exactly prometheus is scraping, and when.

This endpoint also allows for the cloning of any existing metrics in this standalone environment. Input can be both JSON and the [Prometheus Client Exposition Format](#) (which is compatable with the [openmetrics](#) project). In our go application the metrics are implemented using the Prometheus and [Victoria Metrics](#) golang client libraries. We use these to allow simulation of actual metrics gathering for testing.

See the [metrics examples and usage](app/test/examples.sh) for more. The Pingpong app exposes various endpoints like...

```
# the basic endpoints exposed
curl https://alertstack.io/ping

curl https://alertstack.io/metrics

curl https://alertstack.io/create

curl https://alertstack.io/update

curl https://alertstack.io/echo

curl https://alertstack.io/v2/enqueue
```



## A Second Objective

I want to encourage standardisation of the actual alerting and action display templates that format the messages/events sent upstream to services like Pagerduty, Slack and elsewhere. Standardised so the look and key informations look consistant to a responder no matter the service. Some advantages to this standardisation include...

- Greatly simplify the variation and inconsistencies that can occur in field names, message formats (pagerduty, slack, mobile) accross the various projects/yaml resources in a company.
- Reduce the complexity and redundancy of the various rules/alerts type files (env1.Alerts.xml, env2.Alerts.xml...) slowly over time.
- Provide a dynamic and contextually relevant set of **actionable** links for responders: On-Call, SRE, IR, and others.
- Automate insertion of Runbook, Logs, Grafana, Dashboard links with sensible fallbacks for default.
- Reduce the amount of prior knowledge a responder needs to know to address alerted issues. Achieved through the insertion all the relevant key information in the output formats. Like back links to Prometheus, Grafana, Alertmanager, Logs etc.

- Faster mean time to repair [mttr](#) for incidents (also mean time to mitigate [mttm](#)).

## Quickstart

---

This stack should run as-is with the configuration it has. The main two files that are of interest would be the [prometheus/rules.yml](#) file and [alertmanager/alertmanager.yml](#)

### [prometheus/rules.yml](#)

Where we define the fields, rules, and triggers for alerts

### [alertmanager/alertmanager.yml](#)

Define the set of receivers to be notified of an alert and under what circumstances.

## Run it...

---

```
# assuming clone done
# git clone git@gitlab:rondomondo/alertstack.io.git

# create the actual real ssl certs and renew if needed
./letsencrypt/build.sh password

# sanity test - expect to see 2 ssl certs alertstack.io and alertstack.io
docker run --rm --name alertstack-certbot abcdef/alertstack/certbot certbot certific

# build the pingpong service/app and anything else we need may need
./build.sh password

# build anything we need and start up the services (also attach some logging)
UNZIP_PASSWORD=password docker compose up --build -d -V

# sanity test
curl -v -H 'Host: alertstack.io' https://alertstack.io/time

# attach some logs
docker-compose logs --follow

# WE ARE ACTIVE AT THIS STAGE
# Open another terminal window and then do something like
cat app/test/examples.sh

# Create a metric to match a realistic one
echo "$(cat app/test/bgp_neighbors_status.prom)" | curl --data-binary @- https://ale

# Update a metric to match a realistic one
echo "$(cat app/test/bgp_neighbors_status.prom)" | curl --data-binary @- https://ale

# List all the meetrics as per normal
curl 'https://alertstack.io/metrics'

# The WEB UI components are at the following locations
# prometheus
http://alertstack.io:9090
```



```
# grafana (admin/grafana)
http://alertstack.io:3000

# alertmanager
http://alertstack.io:9093

# pingpong
https://alertstack.io/echo

# pagerduty proxy
https://alertstack.io/v2/enqueue
```

Tip: Use the docker compose logs --follow we can monitor all the interaction with the alertstack components. Particurallly useful to see when other systems are calling or when alerts get triggered, as well as seeing the JSON prettified view of slack and pagerduty payloads

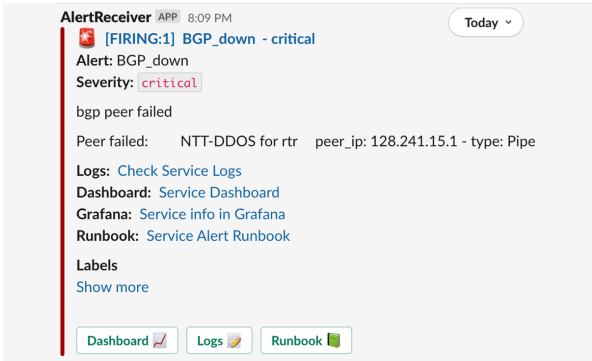
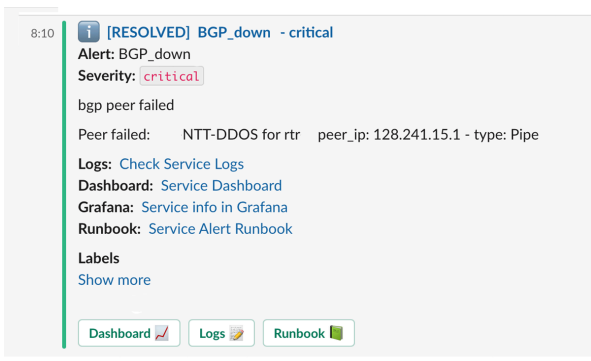
## Aligning Output Template formats for Alerts

Notice we make extensive use of the [Notification Templates](#) for both Prometheus and Alertmanager. These help to format the output in a consistant way accross Pagerduty, Slack and others. The objective is to make them look as similar as possible withing the constraints of the features the relevent integration.

So these are actually [Go Templates](#) that we use to customize our [Prometheus Templates](#) and also the [AlertManager templates](#)

Here is a real example of Alerts sent to PagerDuty and Slack, for the same alert trigger. The key thing to notice is the similar field content and the inclusion of contextually relevant Dashboard, Runbook, Logs links...

Slack Alert Presentation	Slack Alert Presentation
<div><div># alert-receiver ▾</div><div><div><div>🚨 [FIRING:1] proxy_partition_usage_percentage_warn Today ▾</div><div>Alert: proxy_partition_usage_percentage_warn</div><div>Severity: <span>warning</span></div><div>Partition /var/log crossed usage percentage limit on host.docker.internal:8090</div><div>The current percentage <span>2</span> for partition <span>/var/log</span> crosses warning usage percentage limit on <span>host.docker.internal:8090</span></div><div>Logs: <a href="#">Check Service Logs</a></div><div>Dashboard: <a href="#">Service Dashboard</a></div><div>Grafana: <a href="#">Service info in Grafana</a></div><div>Runbook: <a href="#">See Service Runbook</a></div><div>Labels</div><div><a href="#">Show more</a></div></div><div><div>Dashboard</div><div>Logs</div><div>Runbook</div></div></div></div>	<div><div># alert-receiver ▾</div><div><div><div>4:32 🚨 [FIRING:1] proxy_partition_usage_percentage_pager Today ▾</div><div>Alert: proxy_partition_usage_percentage_pager</div><div>Severity: <span>critical</span></div><div>Partition /var/log crossed critical usage percentage limit on host.docker.internal:8090</div><div>The current percentage <span>2</span> for partition <span>/var/log</span> crosses critical usage percentage limit on <span>host.docker.internal:8090</span></div><div>Logs: <a href="#">Check Service Logs</a></div><div>Dashboard: <a href="#">Service Dashboard</a></div><div>Grafana: <a href="#">Service info in Grafana</a></div><div>Runbook: <a href="#">See Service Runbook</a></div><div>Labels</div><div><a href="#">Show more</a></div></div><div><div>Dashboard</div><div>Logs</div><div>Runbook</div></div></div></div>

Slack Alert Presentation	Slack Alert Presentation
	

## Project and file structure layout

This describes the way the project is laid out. You can run any part of this individually.

## Directory structure and other dependents:

```

.
├── compose.yaml
├── grafana
│   └── datasource.yml
├── prometheus
│   ├── rules.yml
│   ├── prometheus.yml
│   └── web.config.file
├── alertmanager
│   ├── alertmanager.yml
│   └── templates
│       ├── default.yml
│       ├── slack.yml
│       ├── slackExtraSRE.yml
│       ├── pagerDuty.yml
│       ├── pagerDutyExtraSRE.yml
│       ├── serviceNow.yml
│       └── serviceNowExtraSRE.yml
├── app
│   ├── build.sh
│   ├── Dockerfile
│   ├── pingpong.go
│   ├── test
│   │   ├── examples.sh
│   │   └── *.prom
├── letsencrypt
│   ├── certs
│   ├── build.sh
│   ├── Dockerfile
│   ├── etc.letsencrypt.tar.zip
│   └── requirements.txt
├── assets
│   ├── screenshots1.jpg
│   ├── ...
│   └── screenshots9.jpg

```



└─ README.md

# Test service in Pagerduty is – create your own as needed  
<https://abcdef.pagerduty.com/service-directory/ABCDEF>

Content  
<https://abcdef.pagerduty.com/incidents/ABCDEF123>

Change Key  
<https://abcdef.pagerduty.com/services/ABCDEF/integrations/ABCDEF123>

# Test Slack App/Channel is  
<https://testplayground-global.slack.com/apps/ABCDEF123-alertreceiver?settings=1>

Content  
<https://testplayground-global.slack.com/archives/ABCDEF/ABCDEF123>

## The Docker *compose.yaml* file

---

```
services:
  prometheus:
    image: prom/prometheus
    networks:
      - default
    container_name: prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--web.enable-admin-api'
      - '--web.enable-lifecycle'
      - '--log.level=debug'
    # - '--web.config.file=/etc/prometheus/web.config.file'
    ports:
      - 9090:9090
    restart: unless-stopped
    volumes:
      - ./prometheus:/etc/prometheus
      - prom_data:/prometheus
  alertmanager:
    image: prom/alertmanager
    networks:
      - default
    container_name: alertmanager
    command:
      - '--config.file=/etc/alertmanager/alertmanager.yml'
      - '--web.external-url=http://alertstack.io:9093'
      - '--log.level=debug'
    ports:
      - 9093:9093
    restart: unless-stopped
    volumes:
      - ./alertmanager:/etc/alertmanager
      - prom_data:/alertmanager
  grafana:
    image: grafana/grafana
    networks:
      - default
    container_name: grafana
    ports:
```



```

    - 3000:3000
restart: unless-stopped
environment:
  - GF_SECURITY_ADMIN_USER=admin
  - GF_SECURITY_ADMIN_PASSWORD=grafana
volumes:
  - ./grafana:/etc/grafana/provisioning/datasources
pingpong:
  image: abcdef/pingpong
  build:
    context: ./app
    dockerfile: ./Dockerfile
  networks:
    - default
  container_name: pingpong
  ports:
    - 8090:8090
    - 443:443
  restart: unless-stopped
  command:
    - '-port=8090'
    - '-port-tls=443'
    - '-cert=certs/alertstack.io.pem'
    - '-key=certs/alertstack.io.key'
  environment:
    - PORT=8090
    - PORT_TLS=443
alertstack-certbot:
  image: abcdef/alertstack/certbot
  build:
    context: ./letsencrypt
    dockerfile: ./Dockerfile
  networks:
    - default
  container_name: alertstack-certbot
volumes:
  prom_data:

```

This compose.yml file defines a stack the four services `prometheus` , `alertmanager` , `grafana` and `pingpong`

When deploying the stack docker-compose maps the default ports for each service to the equivalent ports on the host in order to make it easier the use web interface of each service like so.

- port 9090 [prometheus](#)
- port 9093 [alertmanager](#)
- port 3000 [grafana](#)
- port 8090 [pingpong](#)

## Building the pingpong service. This is the core alertstack service - **pingpong**

[pingpong.go](#) is a golang application. It exposes multiple endpoints on both HTTP and HTTPS. It implements the prometheus client libs as well as the Victoria Metrics golang client libraries.

The main endpoints of interest are to create/update and view any metric you choose to create. It also implements a simple counter `ping_request_count` that increments every time the <https://alertstack.io/ping> endpoint is called. It also exposes a <https://alertstack.io/metrics> endpoint which Prometheus then scrapes. [targets](#)

## Build everything - more/further detail

```
$ ./letsencrypt/build.sh
```



```
#####
```

```
$ ./app/build.sh
```

```
#####
```

```
build: image abcdef/pingpong ...
[+] Building 3.3s (15/15) FINISHED
=> [1/6] FROM docker.io/library/golang:1.20
=> .....
=> [6/6] RUN CGO_ENABLED=0 GOOS=linux go build -o /

=> naming to docker.io/abcdef/pingpong
```

```
built: image abcdef/pingpong
```

```
# to run this container you can...
```

```
docker run -p 8090:8090 -p 443:443 --detach --rm --name pingpong abcdef/pingpong
```

```
#####
```

```
build: ./pingpong locally ...
```

```
built: ./pingpong locally
```

```
Usage of ./pingpong:
```

```
-cert string
    client certificate file (default server.crt)
-disable-insecure
    Disallow self signed certificates. Allowed by default. Insecure, testing use
-disable-tls
    Disable the additional TLS listener which is listening to PORT_TLS by default
-help
    Show help
-key string
    client certificate key file (default server.key)
-port int
    The port to listen for requests (default 8090)
-port-tls int
    The port to listen for TLS requests (default 8443)
```

```
#####
```

```
# to run this standalone app locally do...
```

```
./app/pingpong &
```

```
sleep 3
```

```
# sample endpoints to call
```

```
# /ping will increase the ping_request_count counter by 1
```

```
curl https://alertstack.io/ping
```



```
# /metrics returns the metrics from the prometheus go client built into pingpong
curl https://alertstack.io/metrics

# /time just returns a time string
curl https://alertstack.io/time
```

## Expected result

Look for new docker image called abcdef/pingpong and also a binary at app/pingpong

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
abcdef/pingpong	latest	9624f3117292	16 hours ago	922MB



## Deploy the full stack with docker compose

Run `docker compose up -d` to start the services.

```
$ docker compose up -d
Creating network "prometheus-grafana_default" with the default driver

Creating pingpong      ... done
Creating grafana        ... done
Creating prometheus     ... done
Creating alertmanager   ... done

:: Network prometheus-grafana_default Created
:: Container pingpong      Started
:: Container grafana       Started
:: Container prometheus    Started
:: Container alertmanager  Started
```



## Expected result

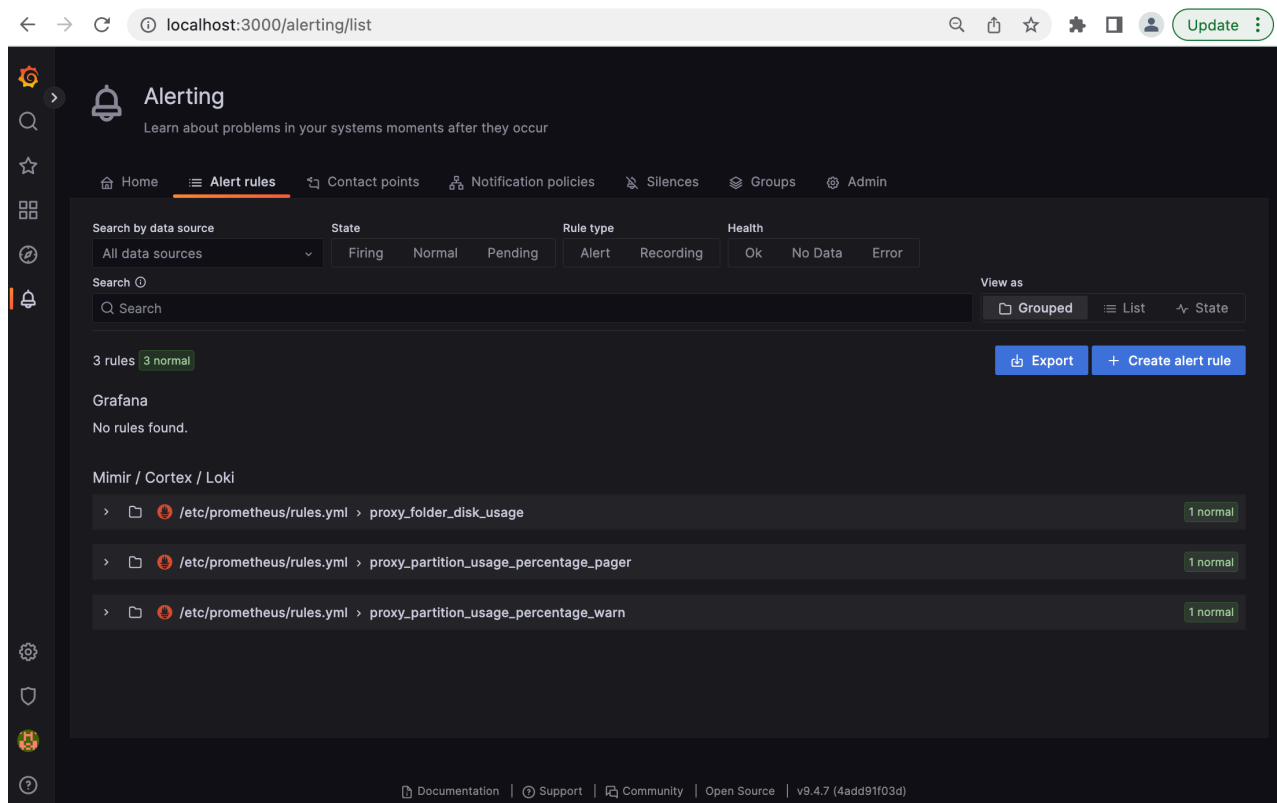
There should be four containers listed with the port mapping as follows

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
c8be35450ed8	prom/prometheus	"/bin/prometheus --c..."	7 minutes ago	Up 7
7a64389d4605	grafana/grafana	"/run.sh"	7 minutes ago	Up 7
dce1d379951b	prom/alertmanager	"/bin/alertmanager -..."	7 minutes ago	Up 7
d56d302b3ede	abcdef/pingpong	"/bin/sh -c '/server..."	7 minutes ago	Up 7



If this is all good then using the Grafana Login credentials from the [compose.yml](#) file you check if [Grafana](#) came up ok, and it looks something like this



## Next steps

Open a terminal window somewhere and kick off some logging to see what is happening

```
docker-compose logs --follow
```



Send some /ping requests to our pingpong app. This will have the affect of incrementing the internal ping\_request\_counter . You can optionally give it a path to use if you like.

Remember you can observe of all this in the log output.

```
curl https://alertstack.io/ping  
# or
```



```
curl https://alertstack.io/ping?path=/tmp/consul_members  
# or
```

```
curl https://alertstack.io/ping?path=/tmp/consul_members&receiver=slack-receiver-abc
```

Check out the metrics that the pingpong app displays. This is the same endpoint that prometheus will scrape

```
# view metrics
curl --silent https://alertstack.io/metrics

curl --silent https://alertstack.io/metrics | prom2json | jq
```



## RULES and ALERTS - Testing them out

Remembering the purpose of this effort is to make it easier to develop, standardise, create and manage alert and monitoring rules and receivers across the different Observability systems typically in use across companies. This is a key part of streamlining our incident management and reliability efforts.

In this case we are concerned with Prometheus and Alertmanager. Prometheus takes a user defined set of rules that defines alerts [note It is pretty common to see the words rules and alerts used interchangeably]

So in this case we define our test set of rules/alerts in [prometheus/rules.yml](#). Prometheus then makes use of this [rules file](#) in the rules section of the prometheus configuration file called [prometheus/prometheus.yml](#)

### A look at some rules and alerts file pieces.

```
# more above etc

groups:
- name: BGP_down
  rules:
  - alert: BGP_down
    expr: bgp_neighbors_status == 2
    for: 30s
    labels:
      severity: critical
      extra_slack_receipient: "#alert-receiver-sre"
      pagerduty_group: sre
      servicenow_group: sre-sn
      runbook_url: "https://alertstack.io/runbook/ALERT:isp_failure+--BGP+Down"
    annotations:
      summary: "bgp peer failed"
      description: "Peer failed: {{$labels.pop}}-{{$labels.peer_name}} for POP1
      logs_url: "https://alertstack.io/logs/query-new/logs?id=aBcDeFII&page=0"
      grafana_url: "https://alertstack.io/grafana/d/_aBcDik/{{$labels.pagerduty_
      dashboard_url: "https://alertstack.io/grafana/d/_aBcDik/{{$labels.pagerdut
      runbook_base: "https://alertstack.io/runbook/run_book.md#alert-name-"

- name: proxy_disk_usage
  rules:
  - alert: proxy_disk_usage
    expr: node_directory_size_bytes > 500000
    for: 15s
    labels:
      severity: critical
```



```
extra_slack_receipient: "#alert-receiver-sre"
annotations:
  summary: "Folder {{ $labels.path }} crossed disk usage limit on {{ $labels."
  description: "The byte usage limit for the folder `{{ $labels.path }}` has
  logs_url: "https://alertstack.io/logs/query-new/logs?id=aBcDeFII&page=0"
  grafana_url: "https://alertstack.io/grafana/d/_aBcDik/proxy-disk-usage?vie
  runbook_base: "https://alertstack.io/runbook/run_book.md#alert-name-"
  dashboard_url: "https://alertstack.io/grafana/d/_aBcDik/proxy-disk-usage?v
```

## Some other useful tools and utilities for this type of thing

---

### amtool - alertmanager tool

```
amtool --alertmanager.url=http://alertstack.io:9093/ config show

# test the routes to see how they would resolve
amtool --alertmanager.url=http://alertstack.io:9093/ config routes

# test some routes with labels
amtool --verbose config routes test --config.file=alertmanager/alertmanager.yml sev

# validate rules syntaxes
cd abcdef-alerts-staging-config
promtool check rules alerts/alerts_configuration/**/*
```



## Useful links and resources

---

[promtool](#)

[amtool](#)

[jq](#) [yq](#)

[prom2json](#)

[SRE Glossary](#)

[SRE Booklist](#)

[SRE at Google](#)

[Incident Metrics in Google](#)

[SRE Workbook](#) [SRE Workbook](#)