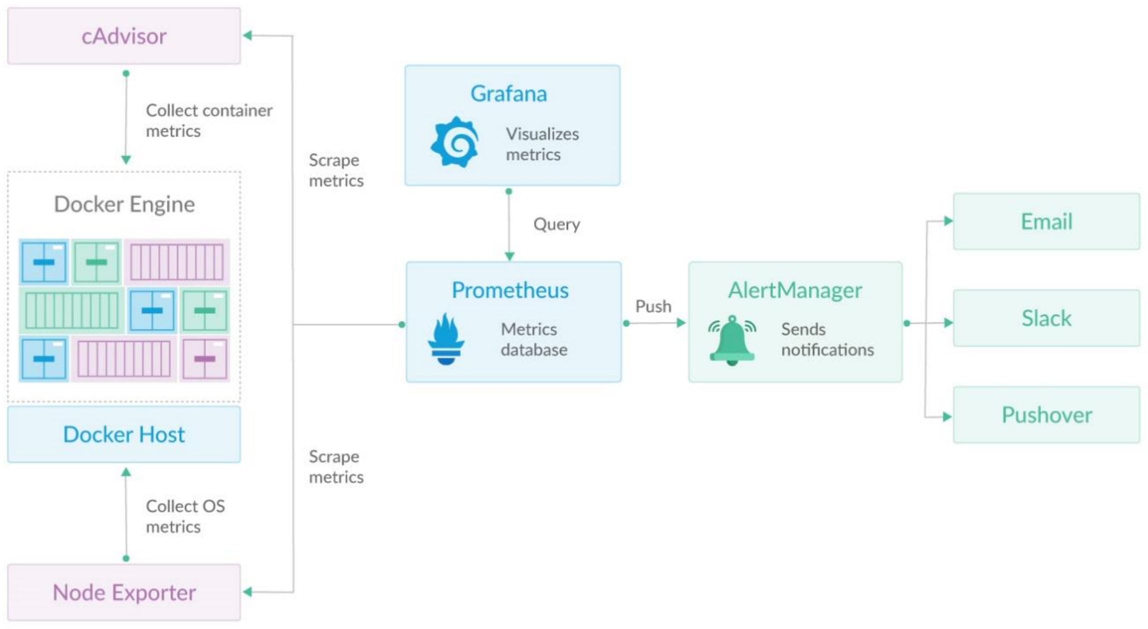
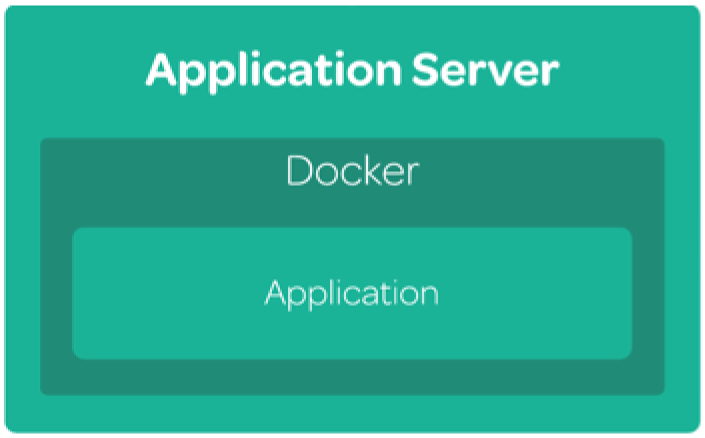
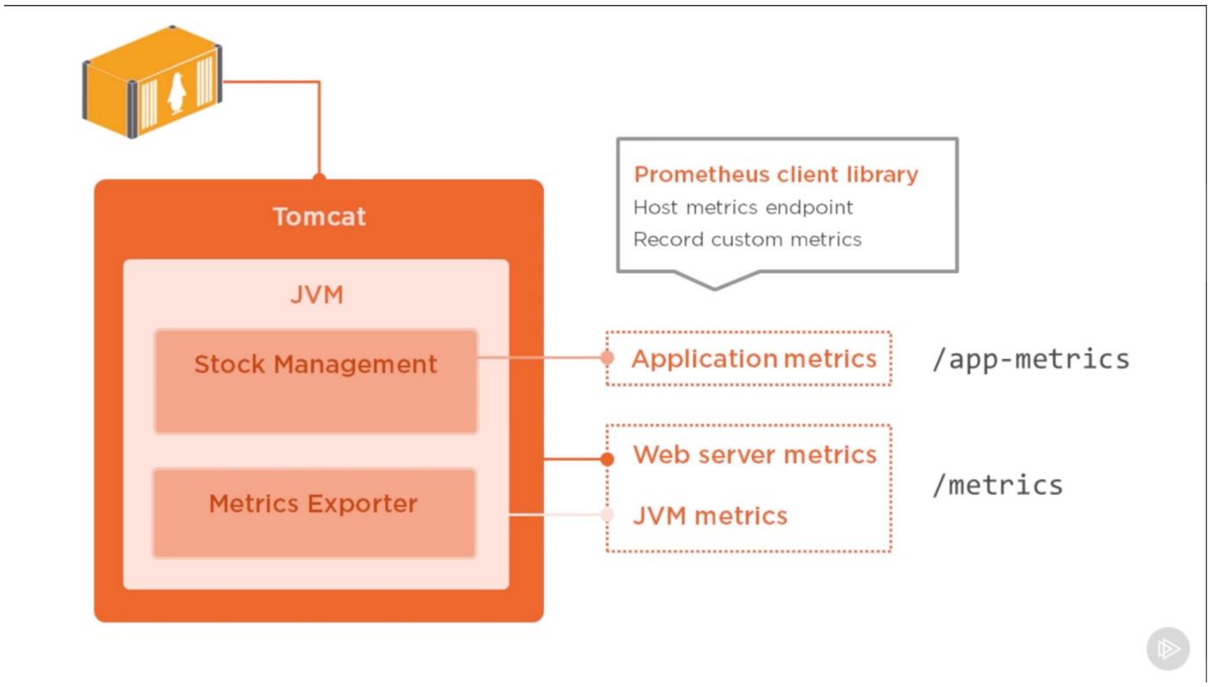
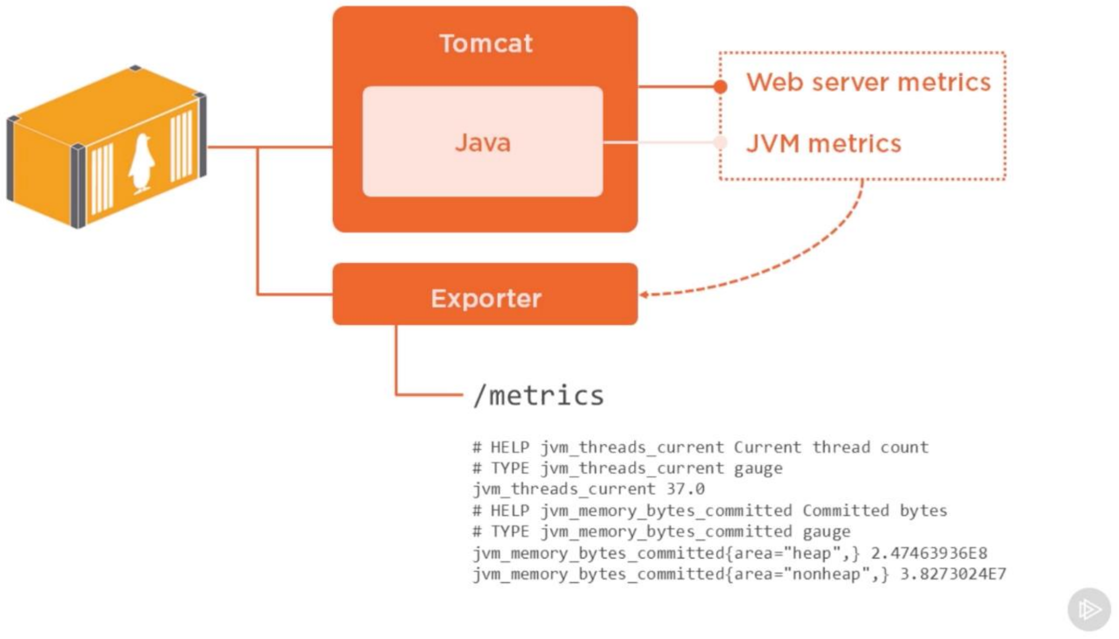
# Prometheus + Grafana with Docker Short Notes

- **Prometheus** - it contains the time series database and the logic of scraping stats from exporters as well as alerts. Prometheus can efficiently manage many vital parameters such as a retention policy or a frequency of metrics collection. It stores data in its own time-series database. Responsible for collecting and storing statistics data
- **Grafana** is to build dashboard to visualize the application which shows key metrics from Prometheus in real-time
- **Alert manager** sends the Prometheus alert to various channels like email, pagers and slack - and so on.
- **Exporters** is node-exporter, it collect system metrics like cpu/memory/storage usage and then it exports it for Prometheus to scrape.
- **Application metrics** - custom metrics you record which are valuable to see in the dashboard, like number of logged-in users or number of checked-out baskets.
- **Runtime metrics** - data already collected by the operating system or runtime host, like the requests per second handled by a web server, or the memory usage.
- **Docker metrics** - metrics from the container platform, including containers running in each state, node availability and health checks.




Application Server
Docker
Application

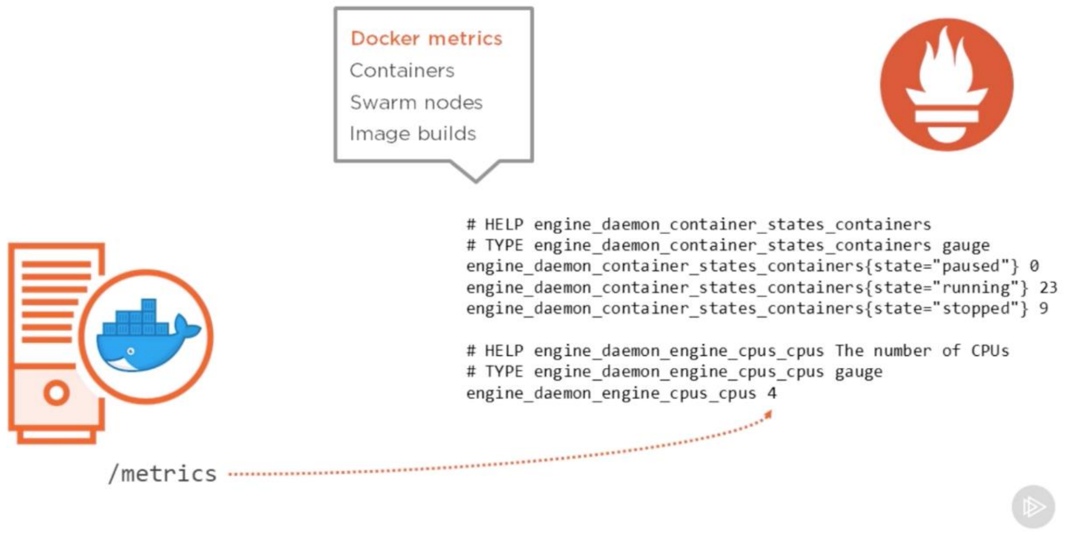

Tomcat
JVM
Stock Management
Metrics Exporter
Prometheus client library
Host metrics endpoint
Record custom metrics
Application metrics
/app-metrics
Web server metrics
JVM metrics
/metrics



```
# HELP jvm_threads_current Current thread count
# TYPE jvm_threads_current gauge
jvm_threads_current 37.0
# HELP jvm_memory_bytes_committed Committed bytes
# TYPE jvm_memory_bytes_committed gauge
jvm_memory_bytes_committed{area="heap",} 2.47463936E8
jvm_memory_bytes_committed{area="nonheap",} 3.8273024E7
```

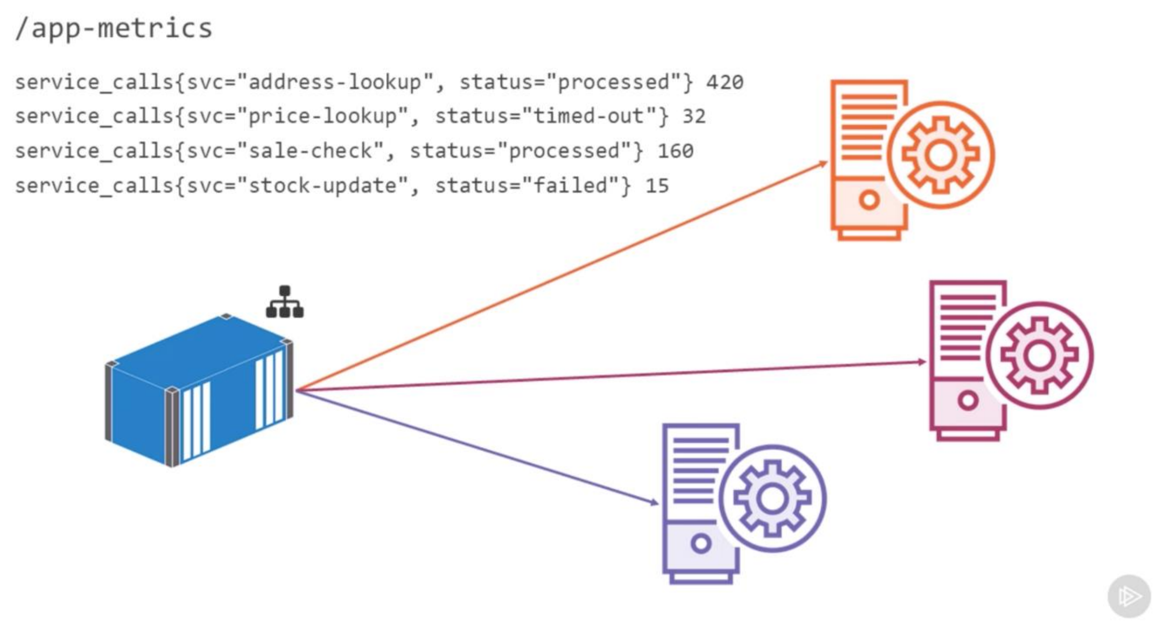
**Container Metrics:**

---

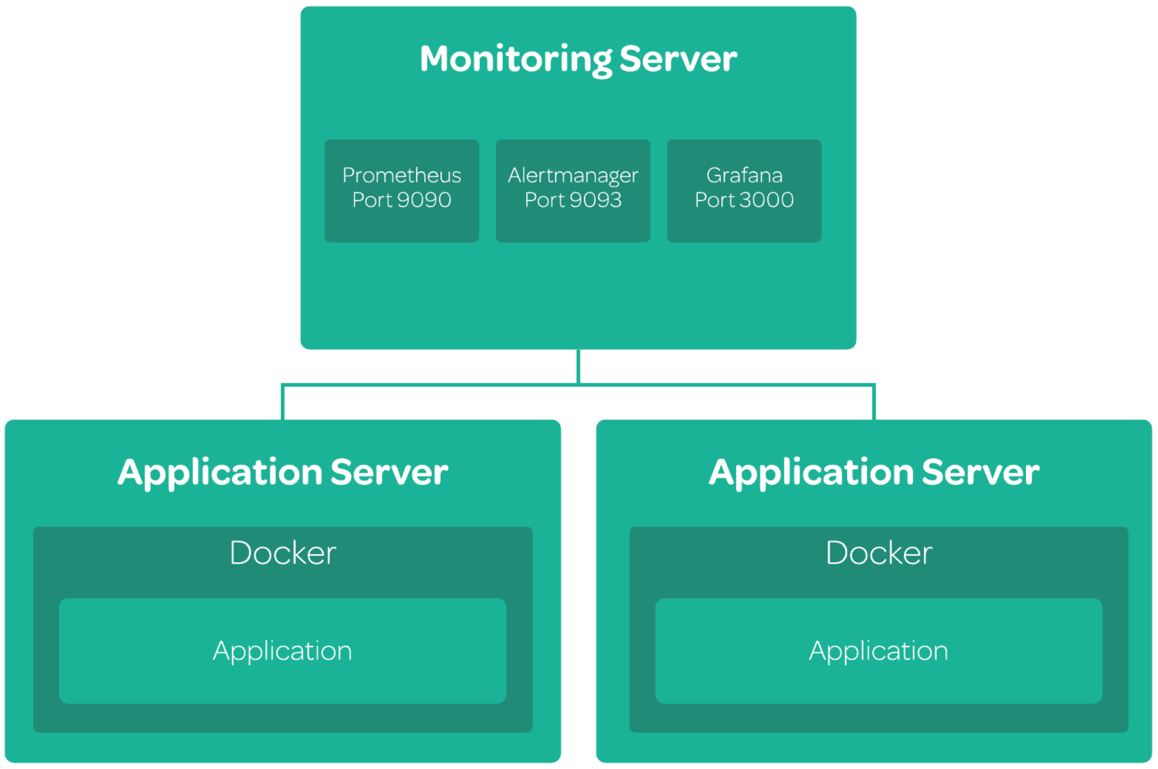


```
# HELP engine_daemon_container_states_containers
# TYPE engine_daemon_container_states_containers gauge
engine_daemon_container_states_containers{state="paused"} 0
engine_daemon_container_states_containers{state="running"} 23
engine_daemon_container_states_containers{state="stopped"} 9

# HELP engine_daemon_engine_cpus_cpus The number of CPUs
# TYPE engine_daemon_engine_cpus_cpus gauge
engine_daemon_engine_cpus_cpus 4
```

**Application Metrics:** Application metrics tell you what's happening inside your containers



```
/app-metrics

service_calls{svc="address-lookup", status="processed"} 420
service_calls{svc="price-lookup", status="timed-out"} 32
service_calls{svc="sale-check", status="processed"} 160
service_calls{svc="stock-update", status="failed"} 15
```



## Access Prometheus from your browser

1. Access Prometheus from your browser:

```
http://[Master Node Public IP address]:9090
```

**Status** > **Targets** will show you more information on the targets in the cluster.

2. You may also access the cAdvisor dashboard at the following address:

```
http://[Worker Node Public IP Address]:8080
```

**Use the provided PromQL queries to interrogate your cluster**

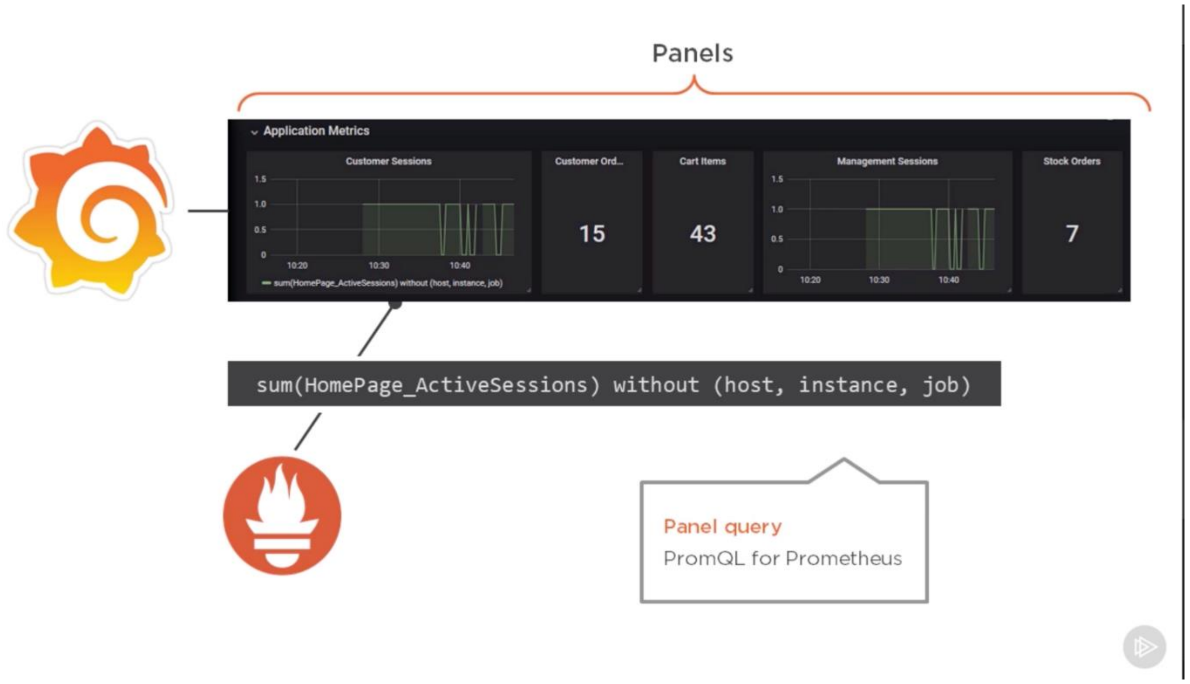
The following are the suggested PromQL queries you may perform.

1. To measure CPU utilization:

```
node_cpu_seconds_total
irate(node_cpu_seconds_total{job="node"}[5m])
avg(irate(node_cpu_seconds_total{job="node"}[5m])) by (instance)
avg(irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
100 - avg(irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
```

2. To measure memory, use:

```
(node_memory_MemTotal_bytes - (node_memory_MemFree_bytes + node_memory_Cached_bytes + node_memory_Buffers_bytes)) / node_memory_MemTotal_bytes * 100
```



**Launching Prometheus**:

```
vi prometheus.yml
```

Launching Prometheus via a very simple docker-compose.yml configuration file

```
scrape_configs:
  - job_name: prometheus
    scrape_interval: 5s
    static_configs:
    - targets:
      - prometheus:9090
      - node-exporter:9100
      - pushgateway:9091
      - cadvisor:8080

  - job_name: docker
    scrape_interval: 5s
    static_configs:
```

```
    - targets:
      - <PRIVATE_IP_ADDRESS>:9323
```

Then, save and quit.

```
:wq
```

**Docker Compose**

The next step is to open our docker-compose file and add three new services. Open docker-compose.yml.

```
vi ~/docker-compose.yml
```

Change the contents of the file to the following:

```
version: '3'
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - 9090:9090
    command:
      - --config.file=/etc/prometheus/prometheus.yml
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
    depends_on:
      - cadvisor
  cadvisor:
    image: google/cadvisor:latest
    container_name: cadvisor
    ports:
      - 8080:8080
    volumes:
```

```
    - /:/rootfs:ro
    - /var/run:/var/run:rw
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
pushgateway:
  image: prom/pushgateway
  container_name: pushgateway
  ports:
    - 9091:9091
node-exporter:
  image: prom/node-exporter:latest
  container_name: node-exporter
  restart: unless-stopped
  expose:
    - 9100
grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - 3000:3000
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=password
  depends_on:
    - prometheus
    - cadvisor
```

Then, save and quit.

```
:wq
```

Next, let's apply our changes and rebuild the environment.

```
docker-compose up -d
```

Then, let's make sure everything is running.

```
docker ps
```