weaveworks

Monitoring Cloud-Native Applications

WHITEPAPER

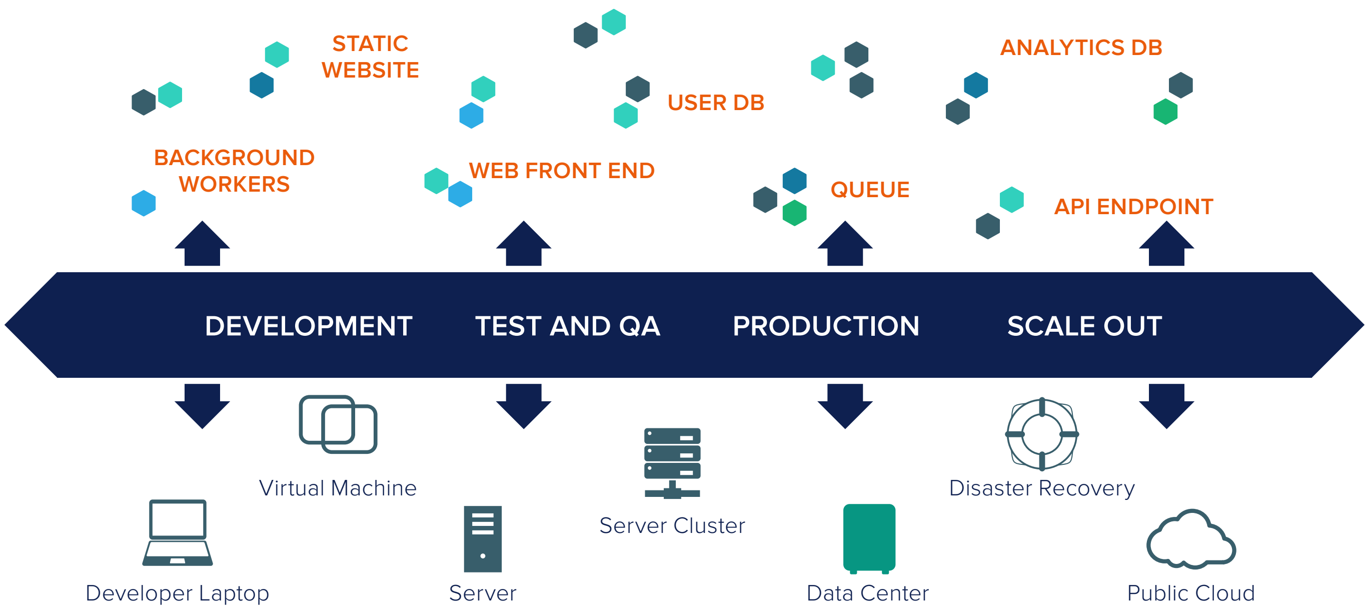
In the process of running your application as a set of microservices, you've discovered that your apps run faster and more reliably. You've also discovered that containerizing your app makes it portable across different cloud environments and that you can run the same container without having to change a single line of app code.

So you've launched your containerized app in the cloud and you want your app to stay up (even if a single instance or your infrastructure fails) and to scale up incoming requests (and likewise scale down when not needed). How do you know that this is happening? That's where monitoring your container-native environment comes in. By tracking specific metrics, you can be alerted as to whether your app is up or down, whether it is scaling correctly, and what corrective action(s) to take.

Monitoring may seem to be only about tracking some key performance indicators (KPIs) on your system to determine if the performance of your microservices are achieving their goals. While this may be true for monolithic apps, monitoring microservices in the cloud is totally different animal.

It's not only about tracking KPIs; it's about tracking the state of dozens (if not hundreds) of containers that are created and destroyed every second. It also involved monitoring how your infrastructure is performing as nodes and pods are automatically scaled up and down. To monitor these types of dynamic environments effectively, you'll need specific tools such as Prometheus, Weave Cortex, or Weave Cloud that are built for container-native environments and that can send alerts when your system is in trouble.

The Challenge



In this white paper, you'll learn:

1 How monitoring a cloud-native environment is different.

2 Why different methodologies, metrics and approaches must be used to effectively monitor microservices (Pro-active Monitoring, RED method, User- Centric Alerting).

3 The concept of Observability and how Monitoring fits into that.

4 Why Prometheus is the best monitoring tool for container-native environments.

5 How Weave Cloud extends Prometheus to provide greater monitoring capabilities in container-native environments.

6 How Weave Cloud gives app developers a better user experience in monitoring their services and infrastructure.

WHY MONITORING?

Monitoring your microservices and infrastructure is critical to troubleshooting problems that come up in the production environment. If you don't troubleshoot your systems quickly, then you risk not only losing time and incurring more operating expenses, but you also risk poor customer satisfaction that can result in lost revenue. The goal of monitoring your microservices and infrastructure is to capture performance bottlenecks and potential issues before they become problems for the dev team or the end user.

To be clear, the goal of monitoring is NOT about just collecting metrics (which are a collection of data inputs that capture a value pertaining to your systems at a specific point in time). But rather, the goal is to convert the metrics you collect into actions you can execute to continuously improve the end-user experience and ROI. These goals are often defined by your customers as service-level agreements (SLAs). Without monitoring, you cannot determine whether your app or service is meeting the SLA agreements.

And in cloud provider environments where your apps or services are running in virtual machines and containers, you need monitoring software to gather metrics that can inform you about the health of your running apps or services as well as its underlying infrastructure. And as you'll learn, Prometheus is one of the best examples of monitoring software for container-native apps running in the cloud.

CHALLENGES OF MONITORING A CONTAINER NATIVE ENVIRONMENT

There are several challenges for monitoring microservices and infrastructure in a container-native environment:

Blackbox versus Whitebox[1]

To know how the app and code is running inside the container, you'll need a monitoring system that is capable of:

BLACKBOX MONITORING
Tests externally visible behavior as a user would see it and is not concerned with what's internally happening, such as tracking the CPU usage of your host machines.

WHITEBOX MONITORING
Monitoring based on metrics exposed by the internals of the system, such as logs or an HTTP handler that emits internal statistics.

Aggregation of metrics[2]

The metrics collected from a single container gives you insight into the behavior of the contents of the container but it does not tell you how the overall microservice or application is performing. To get at this information, you'll need a tool that can analyze at metrics aggregated from all containers that make up a service or application. Examples of aggregated metrics are query response time, URLs that get the most errors, or a service's containers that are exceeding their allocated CPU shares.

1 Monitoring distributed systems by Rob Ewaschuk
2 The Five Principles of Monitoring Microservices by Apurva Dave and Loris Degioanni

Dynamic Environments

The velocity of change in a container-native environment is much greater compared to a virtualized machine environment. Containers for an app or service get created and destroyed every second. Along with that, container orchestration software like Kubernetes dynamically creates and destroys nodes, pods, and replicas to scale with the needs of your service or app or to "self-heal" any of these components that have failed.

The challenge of tracking hundreds (if not thousands) of ephemeral containers running in a dynamic environment demands that your monitoring system be completely compatible with container orchestration software running in the Cloud. Prometheus was designed from the ground up to monitor distributed systems in a dynamic cloud environment. In addition, Prometheus is natively supported by Kubernetes as well. Prometheus and Kubernetes complement each other because both support service discovery and labels. In particular, Prometheus uses labels to identify specific time-series data, which gives you the capability to examine an issue at any point in time.

Why a Time Series Database?

Prometheus is a time-series database monitoring system and is native to a containerized environment. Prometheus is built to monitor applications and microservices running in containers at scale. Data that Prometheus scrapes from running services is a **time-based data type** is queried via the PromQL language. One of the advantages of querying time-based data with PromQL is the ability to step back through time and diagnose a problem in situ without having to independently recreate the issue.

In addition, Prometheus provides these critical monitoring features:

- Whitebox and blackbox monitoring of containers.
- Pulls metrics from the containers through its service discovery mechanism.
- Automatically scales its monitoring based on what the system is doing.
- Aggregates metrics from multiple containers.
- Saves metric data as a time series , which can be queried on using PromQL, Prometheus' powerful query language.
- Send automatic alerts to you in Slack, or email when thresholds and other limitations have been breached.

PRO-ACTIVE MONITORING IS KEY

How you approach monitoring has a big impact on how quickly and efficiently you solve issues and most importantly, preventing them from occurring in your system.

One approach is to simply wait for the customer to tell you there's an issue with your services or application. There's no overhead on your system for monitoring, however, the customer experience is not optimal and it could cause you to lose customers and revenue. This approach is purely reactive.

But what if you monitored for the availability of an app or service (whether it's running or not)? This capability allows you to send out "alerts" as to when a service or app is not available. Definitely better than not monitoring at all.

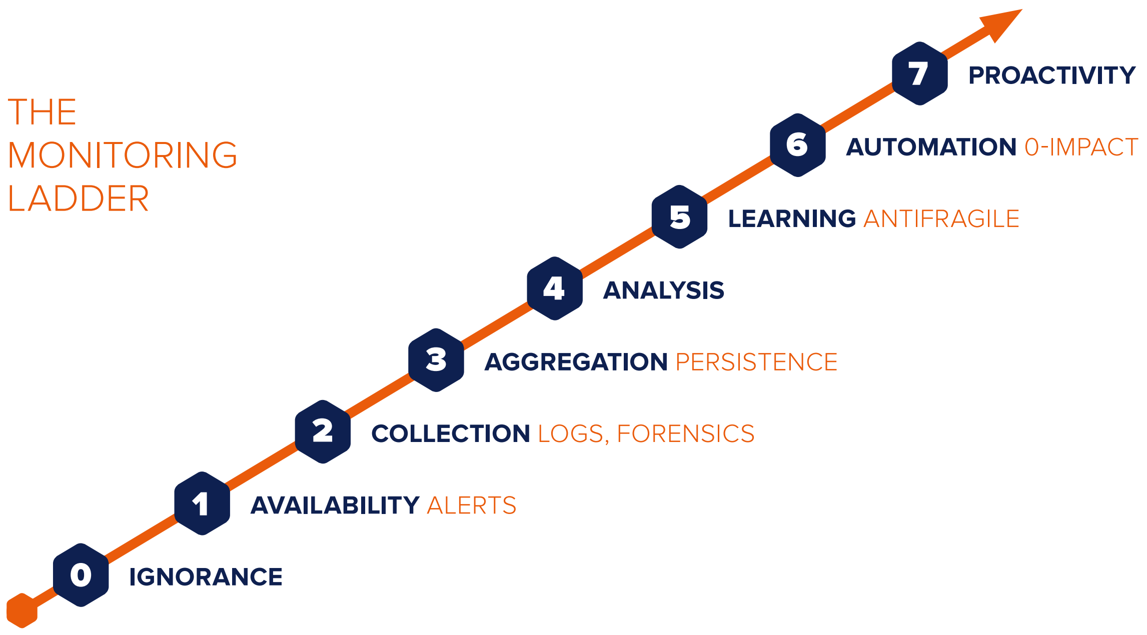
Monitoring for availability alone doesn't tell you why a service or app is up or down. To get at the root cause of a problem, you'll need to monitor for other information, such as the logs for an instance of a service and then aggregate those logs to get an overall picture of the health of your system.

With the aggregated information, you can start correlating the data to pinpoint the source of the problem. This makes it possible to propose fixes or improvements to the code.

You can take a more proactive stance by automating the remedial actions to take when a particular issue has been identified. The goal of automation is to ensure the continuity of the service, that it does not go down even if something is going wrong.

Even better is to use "resiliency" tools (such as the Chaos Monkey service) to attack your own service. The idea is to detect and prevent issues with your system before the services or apps are deployed in production.

All of these additional monitoring capabilities constitute a pro-active monitoring approach that Weaveworks calls the **"Monitoring Maturity Ladder"** model:



METHODOLOGIES FOR MONITORING CONTAINER-NATIVE SERVICES

Weaveworks recommends that you use the the RED method and the USE method to monitor your services and infrastructure.

The RED method helps you monitor the user impact of your microservices; whereas USE method focuses on monitoring the resources that a microservice uses. In this section, we'll look at how each method has its strengths and limitations, and how they complement each other.

RED Method

RED stands for "**R**ate, **E**rrors, and **D**uration". These are the standard metrics that Weaveworks recommends you use for monitoring each microservice that runs. Weaveworks developed the **RED method** as the most efficient way to standardize your metrics for each microservice so you can troubleshoot and remedy issues quickly. NOTE: The RED method is based on the **Four Golden Signals** that Google developed.

Here's the description of each metric in the RED method:

- Rate is the number of requests per second your services are serving.
- Errors is the number of failed requests per second. This rate is expressed as a proportion of the request rate.
- Duration is the distributions of the amount of time each request takes.

So why use the same metrics to measure every microservice? You might think every microservice is different and requires different metrics to track them. But using standard metrics gives your Operations teams the ability to scale up the numbers of services they can monitor.

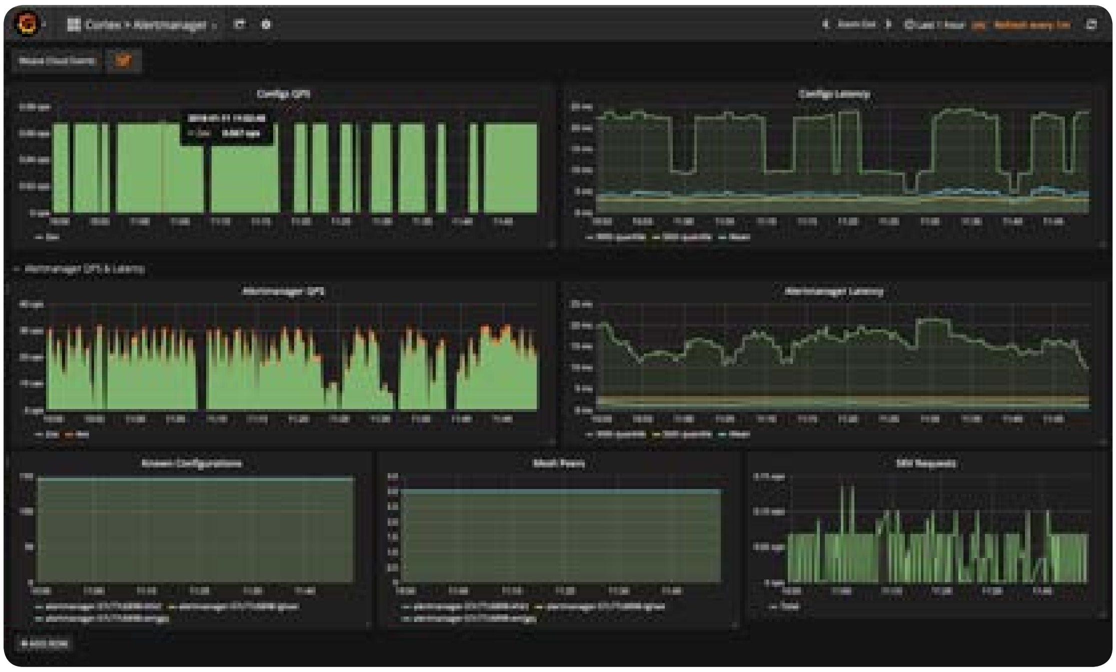
The benefits of standardizing the metrics for every service are:

- **Reduces the service-specific training that the team needs.**
- **Reduces the service-specific info that the on-call team needs to remember for high-pressure incidents.**
- **Lets you automate more of the common repetitive tasks.**

To display these metrics, Weaveworks recommends using **Grafana**, an open source metric analytics and visualization suite for visualizing time series data for infrastructure and application analytics. According to Weaveworks best practices, you could set up your Grafana dashboard with :

- One row per service.
- Two columns with request and error rate on the left and latency on the right. Helps you display info in dashboards that makes it easier to read for troubleshooting.

The limitations of the RED method is that it's mainly used for request-driven services. It's not for batch-oriented services or streaming services.


Grafana dashboard

USE method

This method analyzes the performance of a system. In cloud computing environments, the USE method is suited for determining performance issues, errors or system bottlenecks.

The definition of USE is “For every resource, check the Utilization, Saturation, and Errors”

Where:

> **Resource** is all the physical server functional components (CPU, memory, network interfaces, I/O of storage devices and their capacity, controllers.
>
> **Utilization** is the average time that the resource was busy servicing work or the proportion of the resource used (such as 100 percent utilization means no more work can be accepted). Expressed as percent over a time interval.
>
> **Saturation** is the degree to which the resource has extra work that it cannot service (i.e. handle) and the work is often queued. Expressed as the length of a queue.
>
> **Errors** is the count of error events. Expressed as a scalar value, which means for each unit of time, you have a value.

With the USE method, you create an ordered checklist of metrics to look for. You determine the order of the metrics to look for when troubleshooting and remediating an issue. By defining the ordered checklist, you can quickly know what you did or didn’t check. Similar to how the RED method treats the metrics for all services as the same, the USE method helps you to quickly eliminate root causes of the issue, which helps you focus on the possible subsets of causes. This is very helpful in high-intensity situations where you don’t have much time to respond to an issue.

USER-CENTRIC ALERTING

Having chosen your metrics and instrumented your code for these metrics, how do you know when the service or system is having a negative impact on the user and needs intervention?

By setting up alerts that trigger when a metric exceeds a certain threshold. The alert should not only tell you what's wrong, but it should also tell you what impact the issue has on the user and what action you need to take to remedy it. Weaveworks calls this approach as "User-centric Alerting". This helps on-call DevOps engineers quickly diagnose the issue before customers are affected.

Let's look at an example of how a monitoring tool supports User-centric alerting. Prometheus is a cloud-native monitoring tool that has an "annotation" feature. When creating an alert, annotations allow you to specify extra information for an alert that gives an engineer an idea of how to react to it.

In the following example, the ALERT has an ANNOTATIONS section with fields called **`impact`** and **`description`**:

```
ALERT QueryErrorRate
  IF          job:scope _ request _ errors:rate1m{job="scope/query"}
> 0.1
  FOR         5m
  LABELS      { severity="critical" }
  ANNOTATIONS {
    summary = "scope/query: high error rate",
    impact = "Users experiencing bugs in Weave Cloud Explore",
    description = "The query service has an error rate (response
    code >= 500) of  errors per second.",
    dashboardURL =
  "https://$REDACTED _ INTERNAL _ URL/grafana/dashboard/file/scope-services.
  json",
    playbookURL = "https://$REDACTED _ INTERNAL _ URL/PLAYBOOK.md#query",
}
```

The **`impact`** field gives information about who is impacted by the issue and the description field gives you a detailed explanation of the issue. When you use Weave Cloud-hosted Prometheus, you get built-in "annotations" for an alert (such "No one can log in" or Terminals and other controls are failing for Weave Cloud users"). This helps your DevOps folks quickly remediate the issue. For more information about using annotations, see **"Monitoring Your Kubernetes Infrastructure with Prometheus"**.

MONITORING IS ONE COMPONENT OF OBSERVABILITY

Monitoring is only one technique of an overall approach to help developers and operations diagnose services or app issues. Tracing, logging, and visualization of the services are the other techniques for collecting data that indicate the operational wellness of the service (like error rate, request latency, or queries per second). Together with monitoring, they give “Observability” to the health of your services. This means that a developer makes their apps or service visible observable so that their behavior and impact on users can be monitored.

A system is observable if developers can understand its current state from the outside. Making an application or service observable means they can be in charge of monitoring their app’s behavior and impact on their app’s users. For further details about observability, see Alexis article ‘**GitOps Part 3 - Observability**’.

PROMETHEUS IS THE SOLUTION FOR MONITORING MICROSERVICES

Prometheus is an open source monitoring system that was designed and built from the ground up for monitoring and alerting of container, microservice-based architectures.

Prometheus has these built in features that allow you to retrieve metrics within a container-native environment:

- Pull-based model for obtaining metrics, which means the monitoring system discovers and pulls the metrics from the container instead of pushing metrics
- Collects monitoring data and saves it in a time-series database that can be queried with PromQL.
- Designed for reliability so that if the other parts of the infrastructure are down you can still access the monitoring data
- An alert manager that handles alerts from Prometheus and routes the alerts to the correct receiver

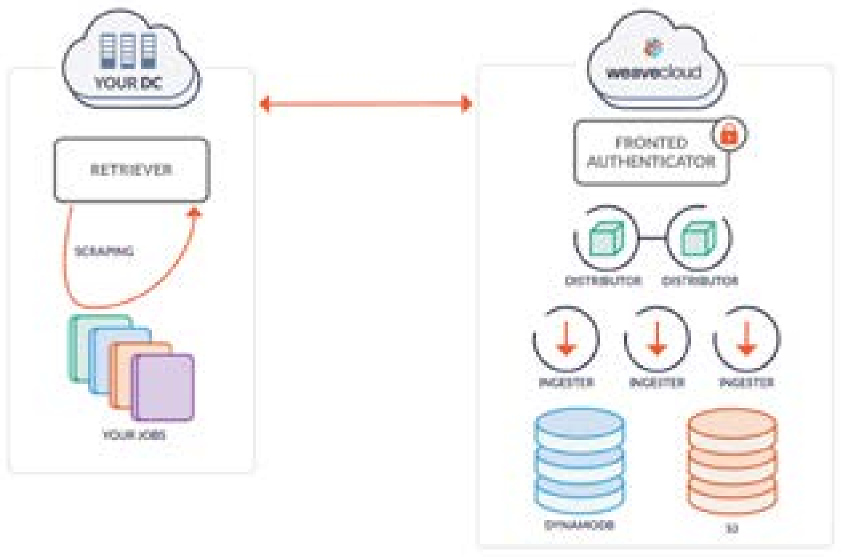
However, there are several limitations to be aware of with the open-source version of Prometheus:

- It runs as a single binary and works on a local, single machine. In other words, you cannot scale Prometheus to have multiple instances of it for a service or application.
- If Prometheus fails on the host, you’ll have to perform manual data retrieval.
- By itself, Prometheus does not offer authentication and access control to its data in the monitoring system. You must build it yourself or provide a 3rd-party alternative.
- There is no built-in storage service for your monitoring data. By default, all data resides on a local disk. To store data externally, you must integrate with a 3rd-party service.

WEAVE CLOUD PROVIDES A BETTER MONITORING EXPERIENCE

Prometheus is an integral part of **Weave Cloud**, a SaaS operations platform for app developers or development teams who are building containerized applications.

PROMETHEUS MONITORING IN WEAVE CLOUD OVERVIEW



Weave Cloud extends Prometheus by providing a distributed, multi-tenant, horizontally scalable version of Prometheus. We host the scraped Prometheus metrics for you, so that you don't have to worry about storage or backups.

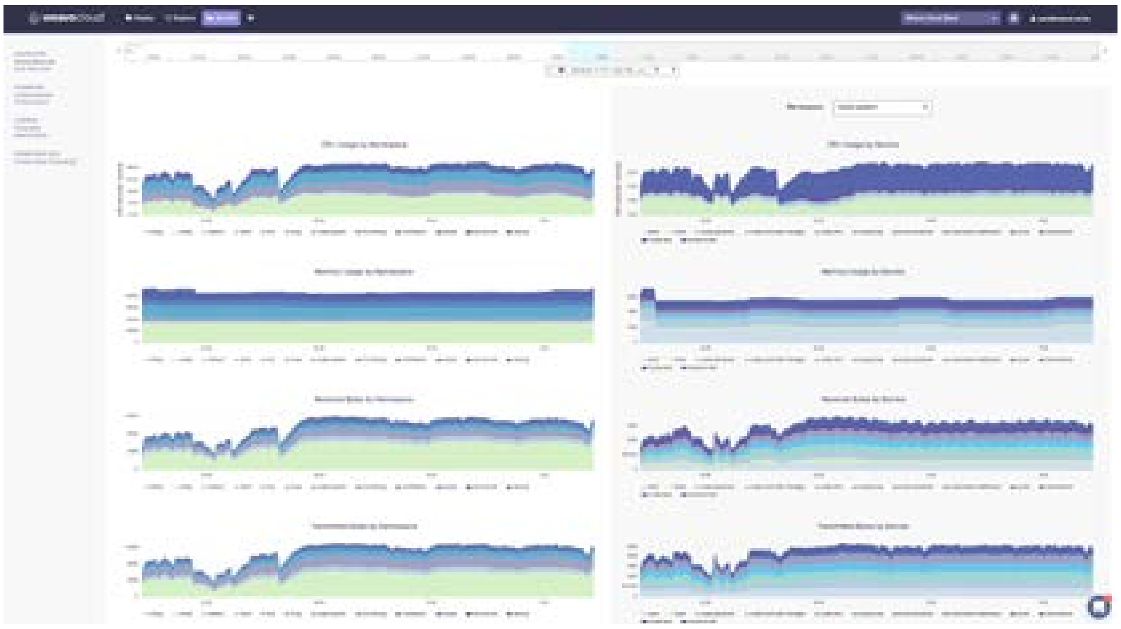
Weave Cloud Monitoring provides the following benefits that the standalone version of Prometheus does not:

- Run on multiple hosts. Without any manual configuration, it can horizontally scale. For example, you can run multiple instances of Prometheus for a large application.
- Prometheus has been rebuilt with a microservices pattern in Weave Cloud. This means there are separate services for each feature such as query, ingest, alerting rules, and recording rules services.
- Provides a replicated set of ingesters so that the failure of one ingester does not mean data loss. The backup ingester automatically handles the data in case of failure.
- Secure access to the monitoring data through a reverse proxy server that is integrated with the Weave Cloud user management service. This provides easy access to the Prometheus and Grafana dashboards for monitoring the data from anywhere on the Internet.
- A built-in storage service that allows you to store time series data from Prometheus. In contrast, the single host version of Prometheus does not automatically provide a storage service.
- A custom user interface for submitting ad hoc Prometheus queries, which includes an auto-complete feature so users can define their desired tracking metrics faster, minimizing the effort for querying time series data.

Weave Cloud is a Complete Solution for Monitoring

The Monitoring feature of Weave Cloud is closely integrated with its continuous delivery (Deployment), and visualization and troubleshooting (Visualization) tools. Weave Cloud provides dashboards for these tools so you can easily act on the data presented by the Monitoring feature. In other words, the integration of Deployment, Visualization, and Monitoring lets you ship features faster and fix problems quicker.

Along with monitoring the current status of your apps and services, you can also easily view historical data with **Weave Cloud's Time Travel** tool and you can check that deployments will not break your running application. This allows you to view the state of your system at any point in time and also allows you to examine how the service or system has changed over time.



Weave Cloud Monitoring Dashboard

CONCLUSION

The important lessons learned about monitoring apps in a container-native environment are:

- Requires tools that are built specifically for monitoring dynamic environments.
- Needs whitebox monitoring to see what's going on inside the containers.
- Needs blackbox monitoring to see how users are affected.
- Use both the RED and USE methods to troubleshoot how your services are performing and how well the infrastructure resources are being used.
- Prometheus is the best tool for monitoring a dynamic container-native environment.
- Weave Cloud extends Prometheus by making it scalable and by providing an easy-to-use UI to perform **queries with PromQL** on the data that Prometheus is pulling and to alert you when any of those data thresholds have been met.

  Try Weave Cloud for FREE