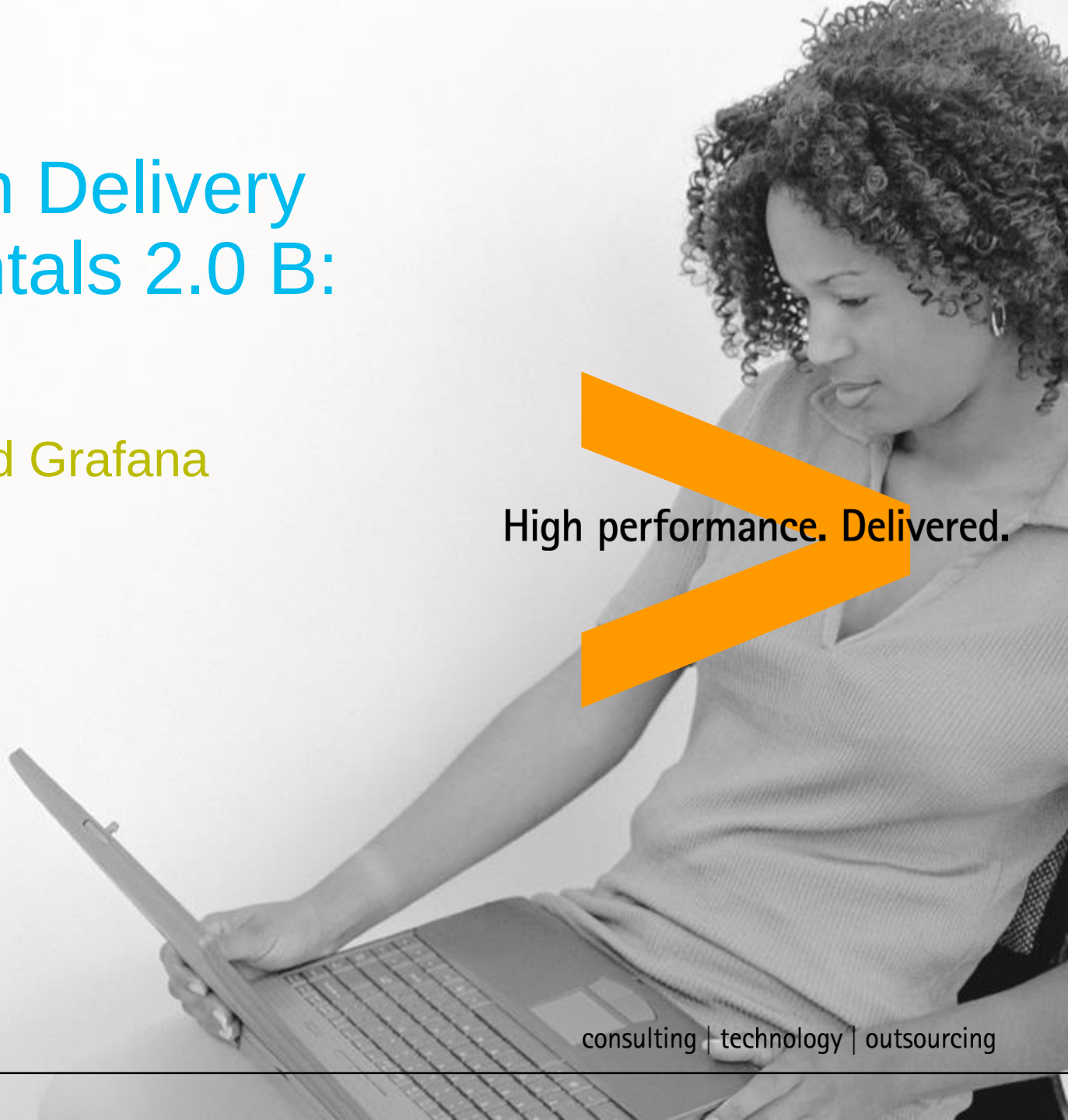
Application Delivery Fundamentals 2.0 B:

Prometheus and Grafana



High performance. Delivered.

consulting | technology | outsourcing

Goals

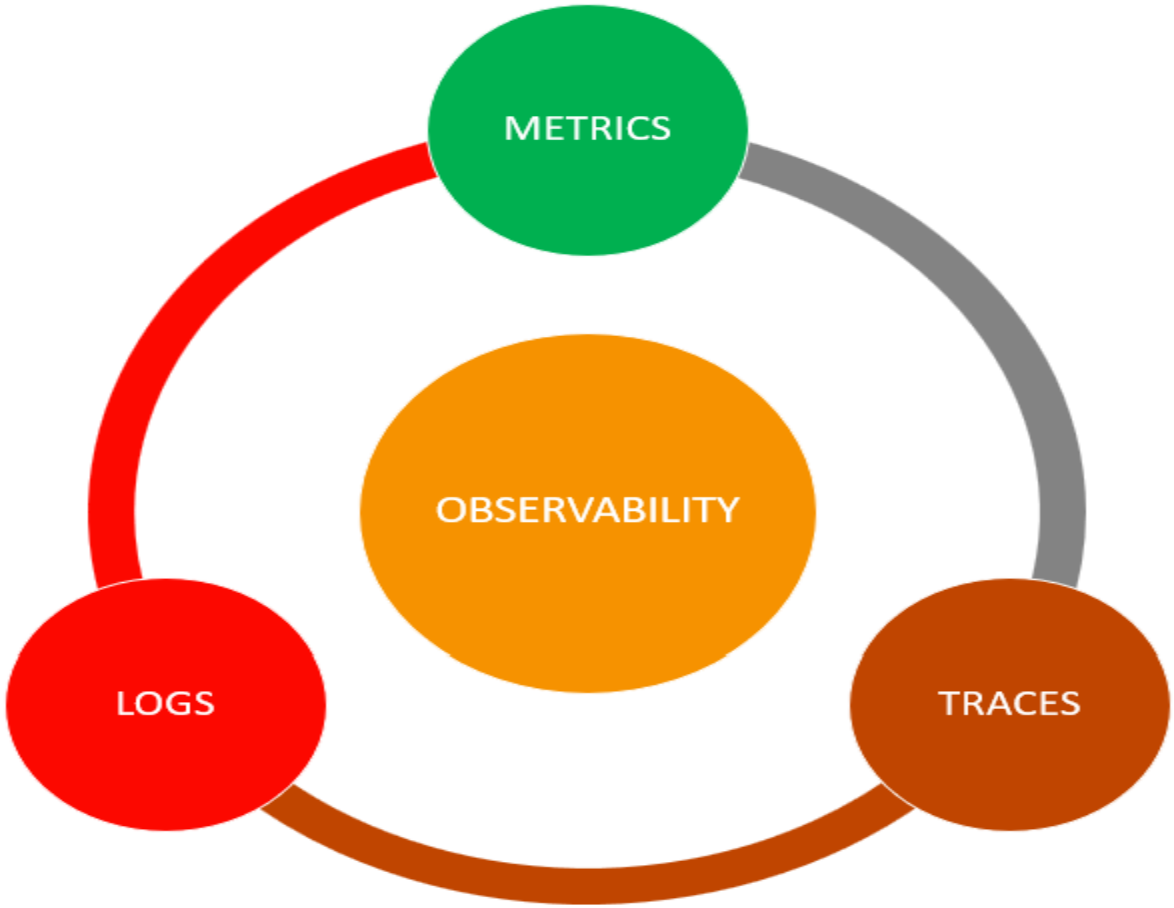
- Observability Concepts
- Prometheus fundamentals
- PromQL
- Instrumentation and Exporters
- Alerting & Dashboarding
- Alerting advanced
- Alert Manager HA
- Prometheus Federation
- Grafana

Observability

- Observability is the measure of how well the internal states of a system can be inferred from knowledge of its external outputs.
- Often confused with monitoring, observability goes a step further by identifying issues in a system and why those issues occur.
- It’s from control theory and has become super crucial in software engineering, especially with the rise of complex, distributed systems and microservices.
- Observability collects data from metrics, logs, and traces, the three pillars, to give a complete view of a system’s health and performance.

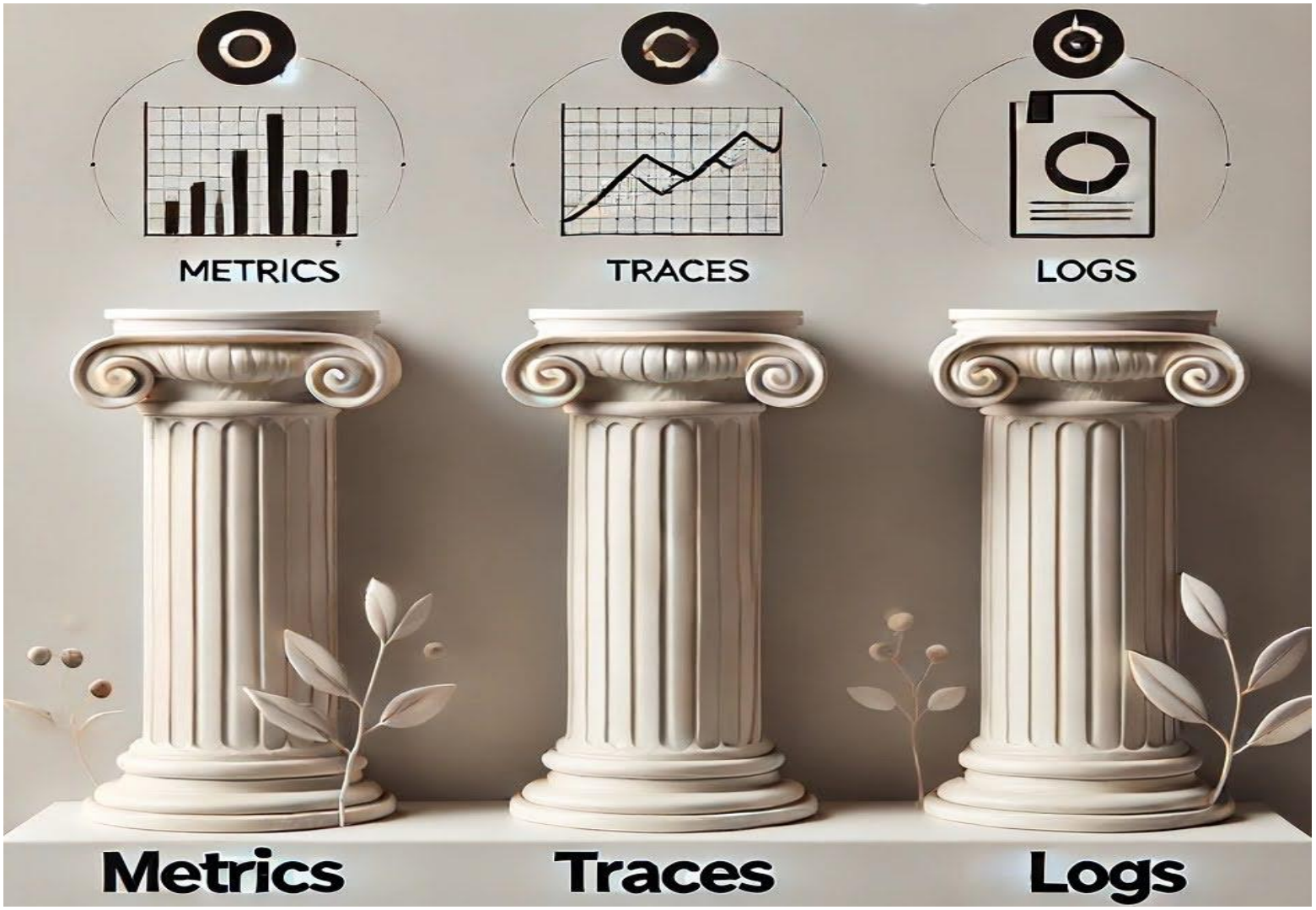
Observability

Microservices



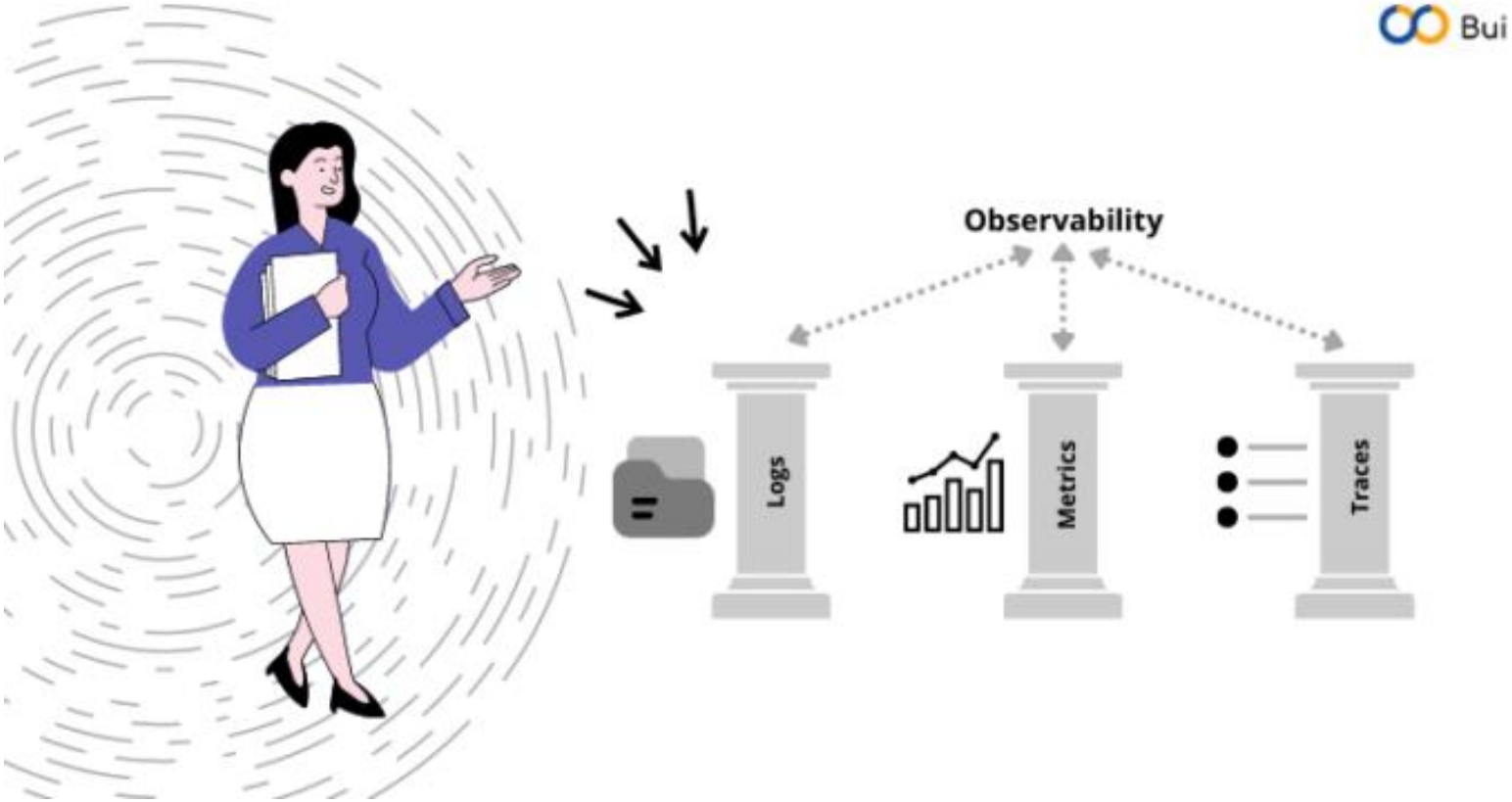
Observability

Microservices



Observability

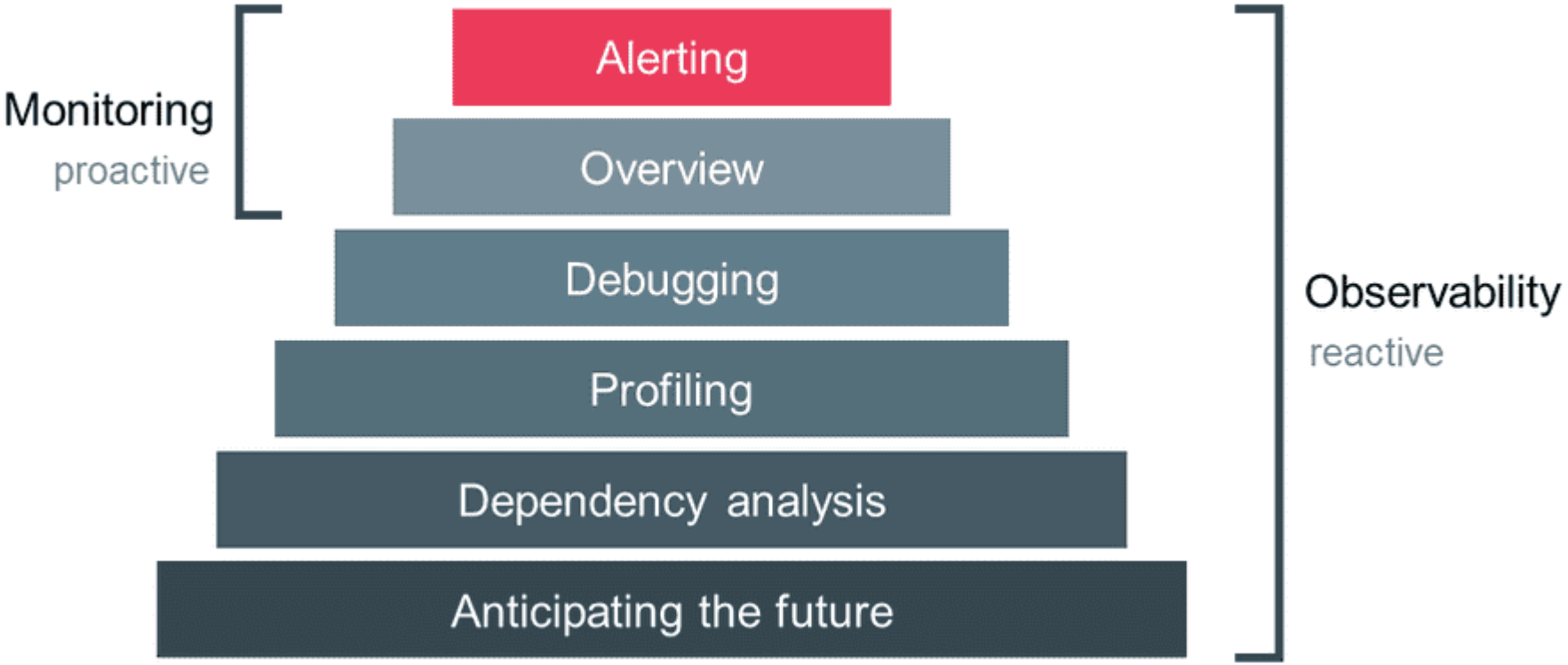
Microservices



Need for Comprehensive Observability

- Less downtime: Faster detection and fixing means reduced downtime and more customer service.
- Better decisions: With visibility into system performance, management can make better decisions on resource allocation and strategic improvements.
- Develop and run faster: Developers and ops teams can catch bottlenecks or failures before they become big problems.
- Better customer experience: Smoother system operation means less customer friction and better service overall.

Observability vs Monitoring



Observability vs Monitoring

Observability

- Tells you why a system is at fault
- Acts as a knowledge base in defining what to monitor
- Focuses on giving context to the data
- Gives a more complete assessment of the overall environment
- Observability is a traversable map
- It gives you complete Information
- Observability creates the potential to monitor different events

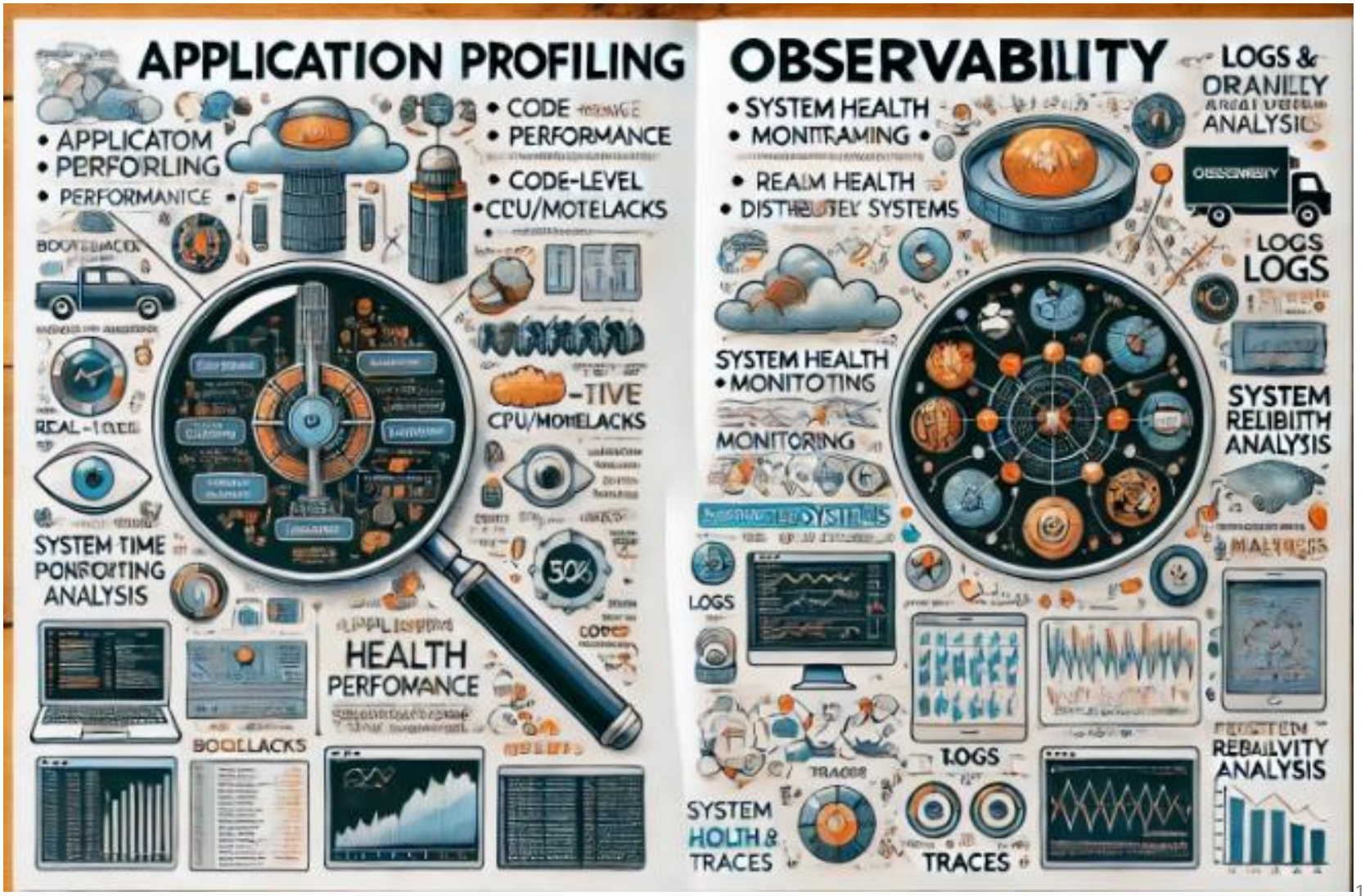
VS

Monitoring

- Notifies you that a system is at fault
- Focuses on monitoring the systems & discovering faults
- Focuses on collecting data
- Focuses on monitoring KPIs
- Monitoring is a single plane
- It gives you limited Information
- Monitoring is the process of using observability

Profiling

Microservices



Profiling

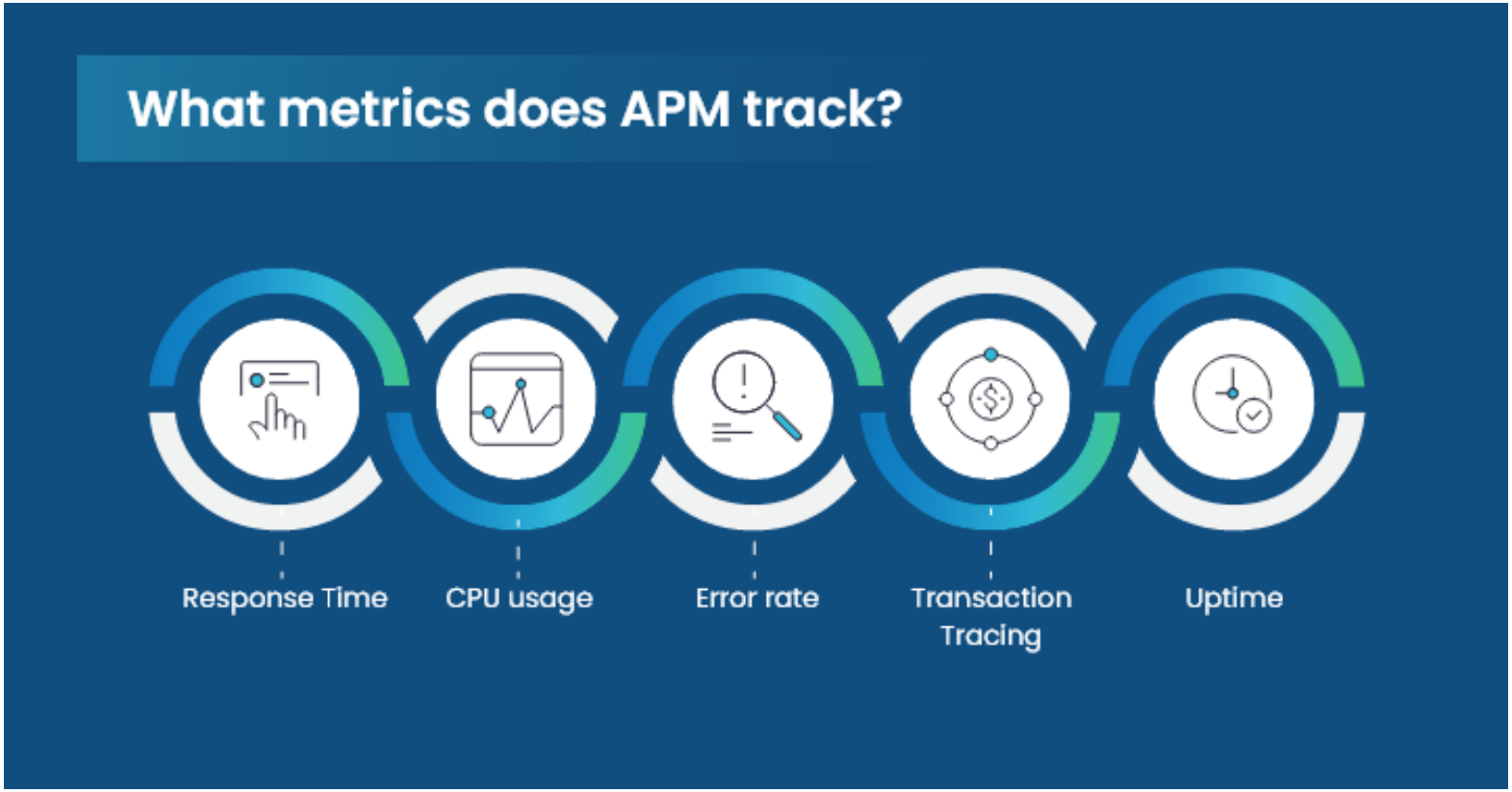
Aspect	Application Profiling	Observability
Focus	Code-level performance and resource usage (CPU, memory, etc.)	System-wide behavior and health (logs, metrics, traces)
Purpose	Identify performance bottlenecks in specific functions or code	Provide real-time monitoring and insights into system health
Granularity	Highly granular, focused on individual threads or functions	Broader scope, focusing on entire applications or services
Timing	Typically done offline or in development	Continuous, real-time monitoring in production environments
Data Type	Low-level data such as CPU cycles, memory usage, and I/O	High-level data like logs, metrics, and distributed traces
Use Case	Debugging, performance optimization, resource tuning	Troubleshooting, detecting issues, and ensuring reliability
Tool Examples	Profilers (e.g., CPU profilers, memory profilers)	Monitoring tools (e.g., Prometheus, Grafana, Datadog)
Scope	Specific code segments or modules	Entire application stack or distributed systems
Insight	Provides detailed insights into code behavior	Provides insights into how the system behaves as a whole
Outcome	Identifies inefficiencies at the code level	Monitors and maintains system performance and availability

Metrics

- Metrics" refers to quantifiable measurements used to track and assess the performance or progress of a specific process, activity, or objective.
- Metrics are essential in various fields, including business, technology, healthcare, and education, to provide objective data for decision-making, analysis, and improvement.
- There are many types of metrics, depending on the context.

Metrics

Microservices



Metrics

Application Performance Management Solution Includes

- Individual Performance web request
- All application usage & performance
- Detailed transaction trace
- Code-level performance profiling
- Basic server metrics
- Application server metrics
- Custom Application metrics
- Application log data
- Application error

Types of Metrics

- **Business Metrics:**
  - **Revenue Growth:** Measures the increase in revenue over a certain period.
  - **Profit Margin:** Percentage of profit generated from total revenue.
  - **Customer Acquisition Cost (CAC):** The cost of acquiring a new customer.
  - **Net Promoter Score (NPS):** Measures customer satisfaction and likelihood to recommend a service/product.
  - **Churn Rate:** The percentage of customers who stop using a service/product.

Types of Metrics

- **Financial Metrics:**
  - **Return on Investment (ROI):** Measures profitability relative to the investment cost.
  - **Gross Profit Margin:** A company's total sales revenue minus its cost of goods sold (COGS).
  - **Debt-to-Equity Ratio:** A measure of a company's financial leverage.

Types of Metrics

- **Project Management Metrics:**
  - **On-time Delivery**: The percentage of projects delivered by the scheduled due date.
  - **Budget Variance**: The difference between the budgeted cost and the actual cost.
  - **Scope Changes**: Number or percentage of changes made to the project scope.

Types of Metrics

- **Technology/Software Development Metrics:**
  - **Velocity:** Measures the amount of work completed during a sprint (agile metric).
  - **Mean Time to Repair (MTTR):** Average time to fix a system after a failure.
  - **Defect Density:** The number of defects found in each amount of code.

Types of Metrics

- **Healthcare Metrics:**
  - **Patient Satisfaction Score:** Measures patient satisfaction with healthcare services.
  - **Average Length of Stay (ALOS):** The average number of days a patient spends in the hospital.
  - **Readmission Rates:** The percentage of patients who return to the hospital after discharge.

Types of Metrics

- **Marketing Metrics:**
  - **Conversion Rate:** The percentage of visitors who take a desired action (e.g., purchasing).
  - **Click-through Rate (CTR):** The percentage of people who clicked on a link after seeing an ad or email.
  - **Cost Per Lead (CPL):** The average cost of acquiring a lead.

Types of Metrics

- **Human Resources Metrics:**
  - **Employee Turnover Rate:** The percentage of employees who leave a company over a given period.
  - **Absenteeism Rate:** Measures employee absences as a percentage of the workforce.
  - **Employee Engagement Score:** A measure of employees' emotional investment in their work.

Importance of Metrics

- Performance Measurement:
  - Metrics allow organizations to gauge how well they are meeting objectives or standards.
  - Decision-Making: With data from metrics, businesses can make informed decisions regarding strategy, resource allocation, and priorities.
  - Benchmarking: Metrics help compare performance over time or against competitors.
  - Continuous Improvement: Metrics enable tracking of progress and identification of areas needing improvement.

Understand logs and events

- Logs and events are essential components in monitoring and analyzing systems, networks, and applications.
- They provide critical insights into the performance, security, and operational status of infrastructure, helping in troubleshooting, auditing, and improving overall system health.
- Understanding the distinction between logs and events and how they are used can help us better interpret system behaviors.

Understand logs and events

Microservices

Understand logs and events

Microservices

Aspect	Logs	Events
Definition	A record of system activities captured over time	A specific action or occurrence within the system
Content	Contains detailed information such as timestamps, messages, errors, etc.	Describes a significant change or state in the system, like an alert or notification
Granularity	Typically continuous and detailed, capturing all system activities	Typically discrete, representing a single instance of change
Purpose	Used for tracking, diagnosing, and understanding system behavior over time	Used to trigger alerts, notifications, or actions based on system state
Use Case	Analyzing system health, debugging, and root cause analysis	Monitoring system state, triggering automation, or handling real-time responses
Storage	Stored as log files or in log management systems	Stored in event logs, metrics, or monitoring systems
Example	"2024-10-11 10:45:32: Error connecting to database."	"User login successful" or "Server down at 10:45 AM"
Relation to Monitoring	Provides context for events, giving detailed system history	Indicates when something of importance occurs
Tool Examples	Logstash, Fluentd, Splunk	Prometheus, Nagios, AWS CloudWatch Events
Time Aspect	Often continuous and spans over a long period	Discrete and represents a point-in-time occurrence

Understand logs and events

- Logs:
  - A log is a detailed, time-stamped record of activity generated by systems, applications, devices, or users.
  - Logs capture various types of information and are used for debugging, auditing, monitoring, and analyzing the state of a system.

Understand logs and events

- **Characteristics of Logs:**
  1. **Time-stamped:** Logs are usually recorded with the exact time and date when the action or event occurred.
  2. **Detailed Information:** Logs typically contain detailed information such as user actions, system processes, requests, errors, and system messages.
  3. **Continuous Generation:** Logs are continuously generated by systems, often creating large amounts of data over time.
  4. **Text-based:** Most logs are stored as plain text or in structured formats like JSON or XML.
  5. **Retained for Auditing:** Logs can be stored for long periods to serve as an audit trail of system activities.

Understand logs and events

- **Types of Logs:**
  - **System Logs:** Captures operating system-related activities such as boot sequences, services starting or stopping, and system errors.
  - **Application Logs:** Records events within an application, such as user requests, database queries, and application errors.
  - **Security Logs:** Includes information about user logins, permissions, access control, and failed login attempts.
  - **Network Logs:** Logs data about network traffic, firewall rules, and packet exchanges.

Understand logs and events

- **Examples of Logs:**
  - Web server access logs capturing incoming HTTP requests.
  - System logs capturing user logins, reboots, or file access.
  - Error logs generated by applications when an error or crash occurs.

Understand logs and events

- **Events:**
  - An event is a specific occurrence or action within a system or application, often defined by triggers or conditions.
  - Events represent key moments that may require attention, such as a security alert, system crash, or significant change in state.

Understand logs and events

- **Characteristics of Events:**
  1. **Discrete Occurrences:** An event is usually tied to a specific action or change, such as a system startup, security breach, or completed transaction.
  2. **Higher-Level Information:** Events tend to summarize important activities, unlike logs which may capture more granular data.
  3. **Trigger-Based:** Events are often triggered by specific conditions or thresholds, such as a system reaching a performance limit, or a user accessing restricted data.
  4. **Alerting and Monitoring:** Events are often tied to monitoring systems that generate alerts, notifications, or reports for administrators.
  5. **Structured Data:** Events are often captured in a more structured way, helping systems and humans process them efficiently.

Understand logs and events

- **Examples of Events:**
  - A user successfully logs in to a system (authentication event).
  - A network intrusion detection system identifies a potential attack (security event).
  - A scheduled backup process successfully completes (system event).
  - An application generates a "transaction completed" event when a purchase is made.

Understand logs and events

- **Differences Between Logs and Events:**
  - **Granularity:** Logs are more granular and detailed, capturing every action or change in a system, while events highlight more significant occurrences.
  - **Purpose:** Logs are used for detailed analysis, diagnostics, and audit trails, whereas events focus on triggering alerts, monitoring system health, and identifying critical issues.
  - **Data Structure:** Logs are often unstructured or semi-structured (e.g., text files), while events tend to be structured and can be more easily parsed by monitoring tools.
  - **Storage:** Logs are often stored for a long time to track past actions, while events are typically used in real-time for immediate responses or alerts.

Understand logs and events

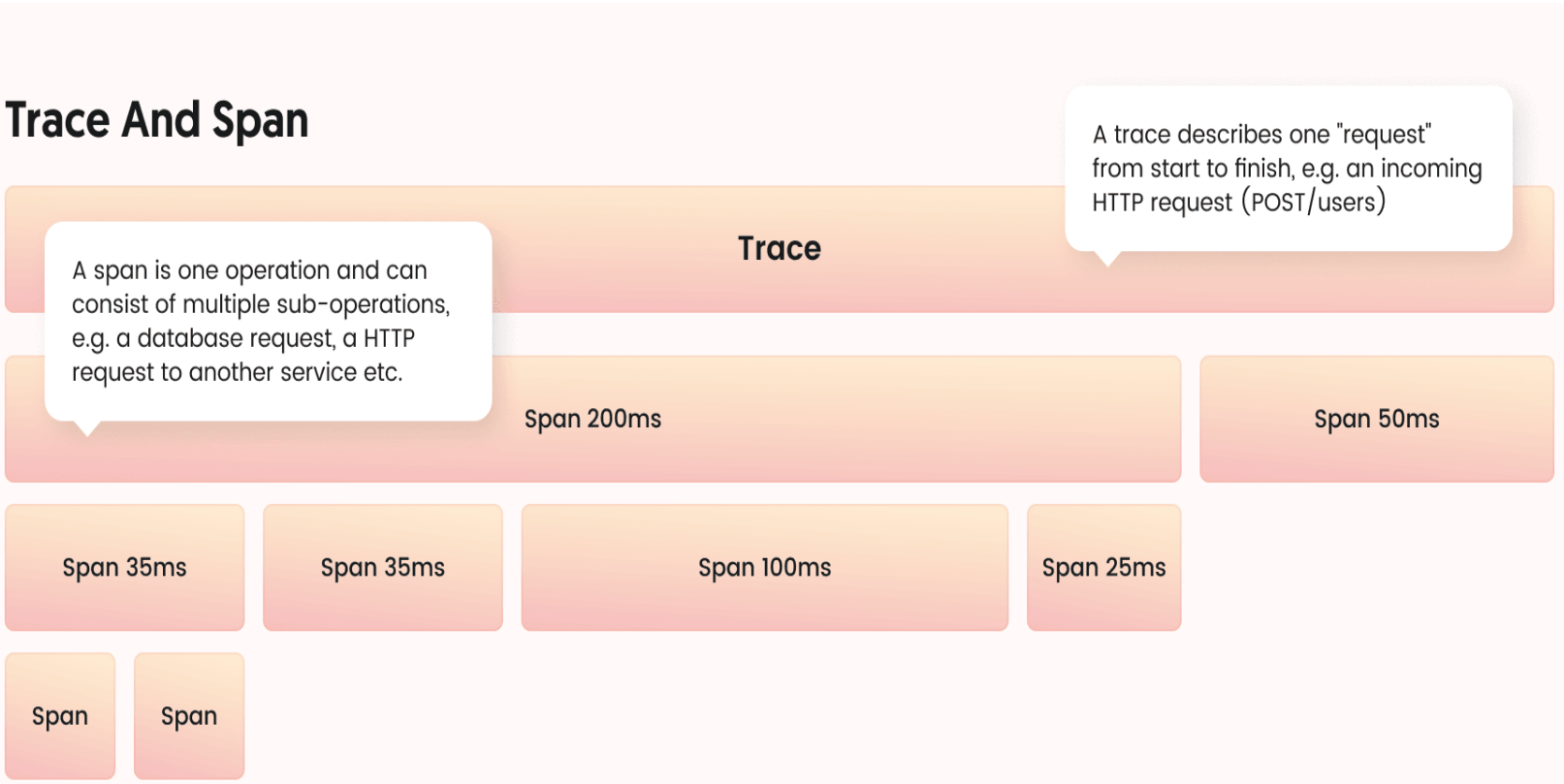
- **Use Cases of Logs and Events:**
  - **Troubleshooting:** When a system issue occurs, logs help pinpoint the exact sequence of actions that led to the issue.
  - For example, an error log in a web server could reveal why a particular page failed to load.
  - **Security Monitoring:** Security logs and events help detect suspicious behavior such as failed login attempts, unauthorized access, or malware activity.
  - Event monitoring tools can raise an alert if certain thresholds are breached (e.g., multiple failed login attempts).

Understand logs and events

- Performance Analysis:
  - Logs can capture metrics such as CPU usage, memory consumption, and network traffic, helping administrators detect performance bottlenecks or inefficient processes.
  - Compliance Auditing: Logs serve as a record of system activity, which is often required for regulatory compliance.
  - Logs can show who accessed data, when, and what actions were taken.
  - Real-Time Monitoring and Alerting: Event management systems (like SIEM or APM tools) continuously monitor systems for events that may require immediate action, such as security breaches or system outages.

Tracing and Spans

Microservices



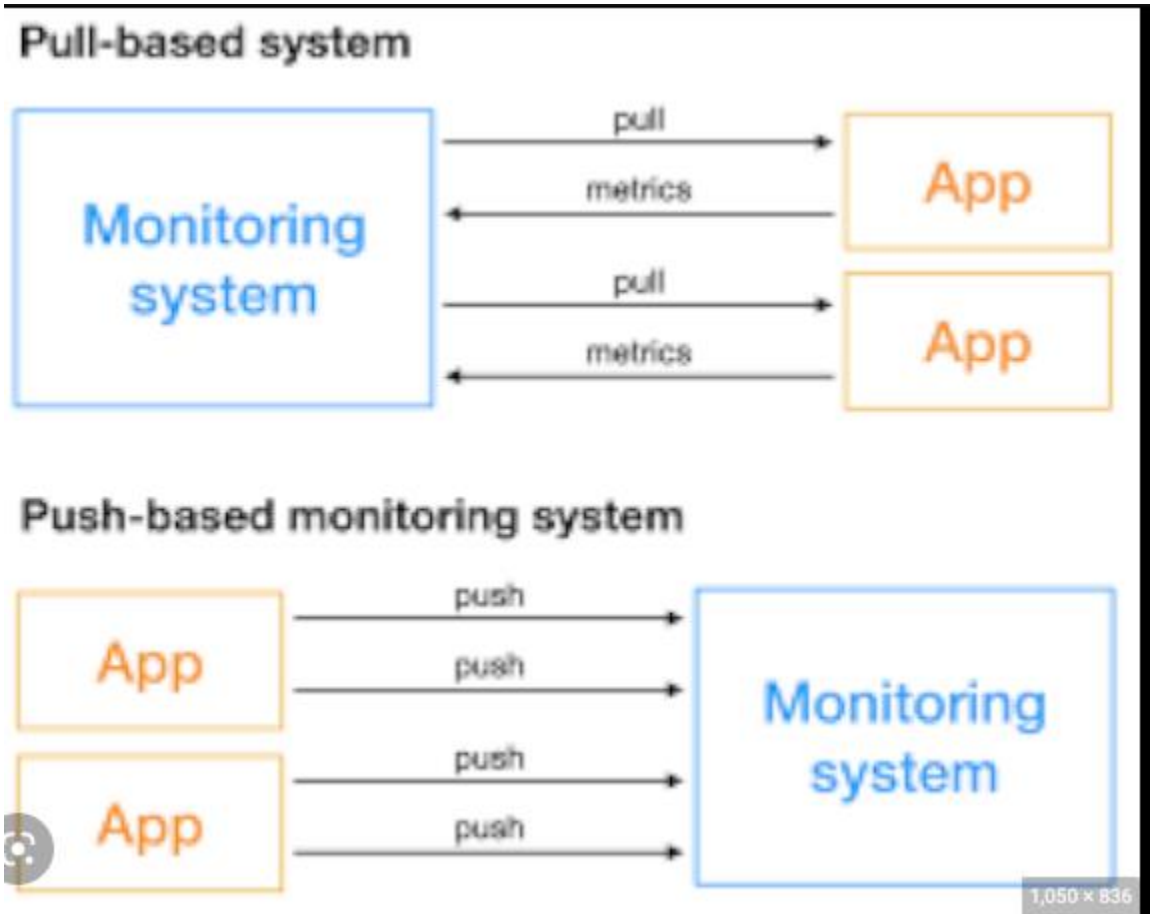
Tracing and Spans

Aspect	Tracing	Spans
Definition	A method used to track the entire journey of a request as it moves through different services in a distributed system.	A single unit of work within a trace, representing a specific operation or step in the request's journey.
Scope	Captures the flow of a request across multiple systems or services, providing a high-level overview.	Represents a specific operation, such as a function call or database query, within the trace.
Granularity	High-level, focused on the end-to-end lifecycle of the request.	Fine-grained, focused on individual components or steps within the trace.
Purpose	To give visibility into how requests traverse through services, helping to identify bottlenecks, failures, or latency issues.	To break down the trace into specific operations, allowing detailed performance analysis for each step.
Hierarchy	A trace consists of multiple spans that represent the steps involved in processing a request.	A span is a segment of a trace, and spans can be nested or linked to show relationships between operations.

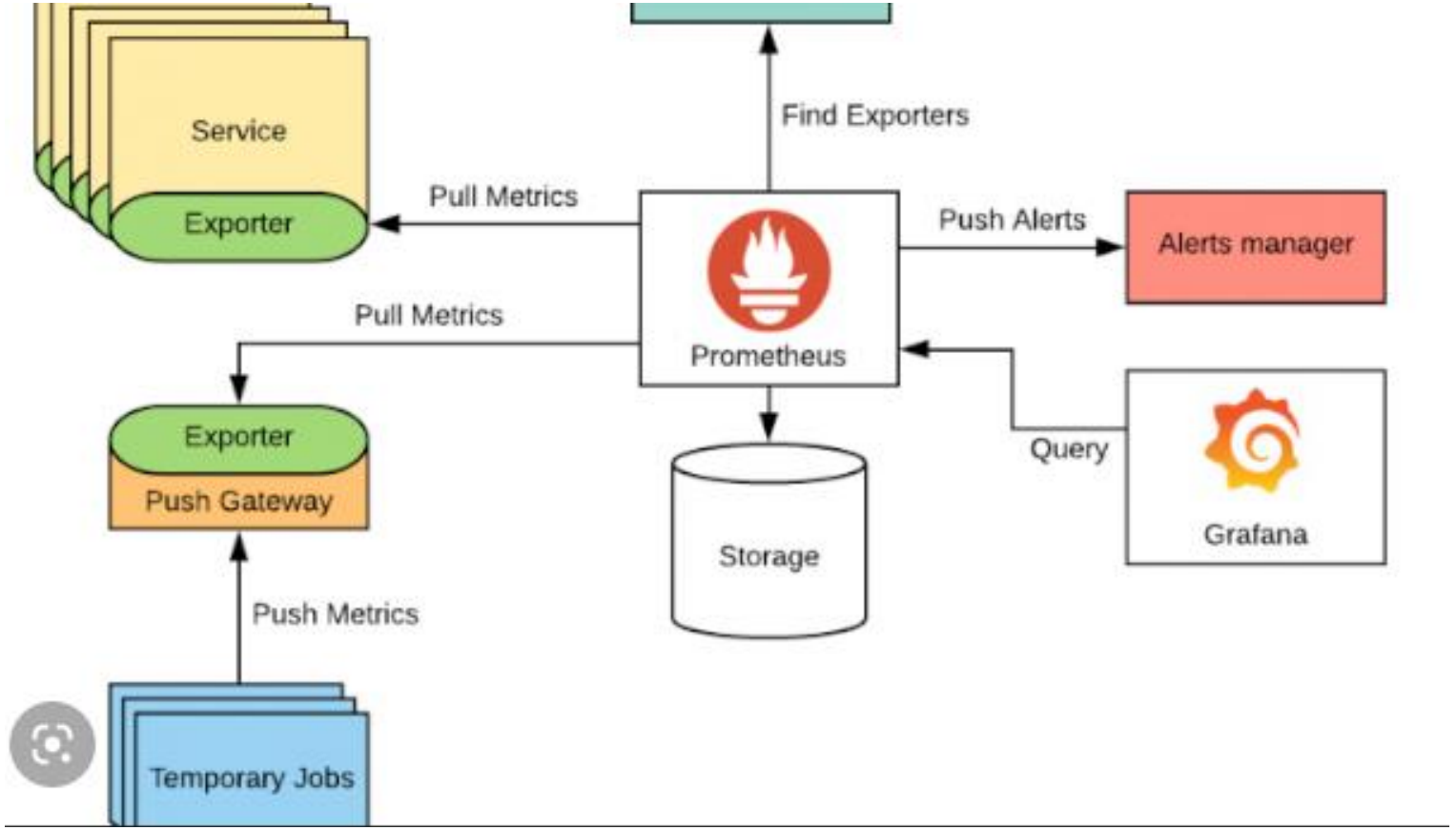
Tracing and Spans

Data Captured	Collects metadata such as trace ID, timestamps, and relationships between spans.	Captures operation-specific details like start/end time, duration, and metadata (tags, logs).
Visualization	Typically visualized as a flow of connected spans, showing the end-to-end path of a request.	Visualized as individual nodes or blocks within the trace, often nested to show dependencies.
Use Case	Helps diagnose latency issues, track request flow, and identify problematic services in distributed architectures.	Helps to pinpoint delays or errors in specific operations within a service.
Example	A trace could show a user's request passing through a web server, an application server, and a database.	A span might represent the database query or the web server processing the request.
Tool Examples	Jaeger, OpenTelemetry, Zipkin	Spans are part of trace data in the same tools (Jaeger, Zipkin).

Push vs Pull


Pull-based system
Monitoring system
pull
metrics
pull
metrics
App
App
Push-based monitoring system
App
App
push
push
push
push
Monitoring system
1,050 × 836

Push vs Pull


Service
Exporter
Pull Metrics
Find Exporters
Push Alerts
Alerts manager
Pull Metrics
Prometheus
Exporter
Push Gateway
Query
Storage
Grafana
Push Metrics
Temporary Jobs

Push vs Pull

- Pull Model (Prometheus's Native Model)
- In the pull model, Prometheus regularly scrapes metrics from targets (services, servers, or applications) at specified intervals.
- It pulls metrics data by sending HTTP requests to a predefined endpoint (usually /metrics).
- How it works:
  - Prometheus server queries the /metrics endpoint of each target.
  - The target exposes metrics in a specific format that Prometheus understands.
  - Prometheus pulls data at intervals specified in its configuration.

Push vs Pull

- Advantages:
  - Prometheus can control the frequency of data collection.
  - It can handle a dynamic list of targets and retry if one fails to respond.
  - Easier to monitor targets behind load balancers since Prometheus pulls from the available endpoint.
  - The target can remain stateless as it doesn’t need to track which systems want its metrics.

Push vs Pull

- Disadvantages:
  - May not work well for short-lived jobs, as these may disappear before Prometheus scrapes their metrics.
  - Not ideal for environments where network access from Prometheus to the target is restricted.

Push vs Pull

- Push Model (Not Native to Prometheus but Achievable)
  - The push model involves the target sending (pushing) its metrics to a central location.
  - Prometheus does not natively support a push mechanism, but the Pushgateway component fills this gap.
  - The Pushgateway acts as an intermediary that receives metrics from applications (especially short-lived jobs) and exposes them to Prometheus.

Push vs Pull

- How it works:
  - Targets push their metrics to a Push gateway.
  - The Push gateway exposes a /metrics endpoint for Prometheus to scrape the pushed data.
  - The Push gateway stores the metrics until Prometheus pulls them.

Push vs Pull

- Advantages:
  - Suitable for short-lived jobs (e.g., batch jobs or CI jobs) that might finish execution before Prometheus can scrape them.
  - Can work in environments where the network setup restricts Prometheus from pulling metrics directly from targets (e.g., firewalls, private networks).
  - Useful when the target cannot keep an HTTP server running to expose metrics continuously.

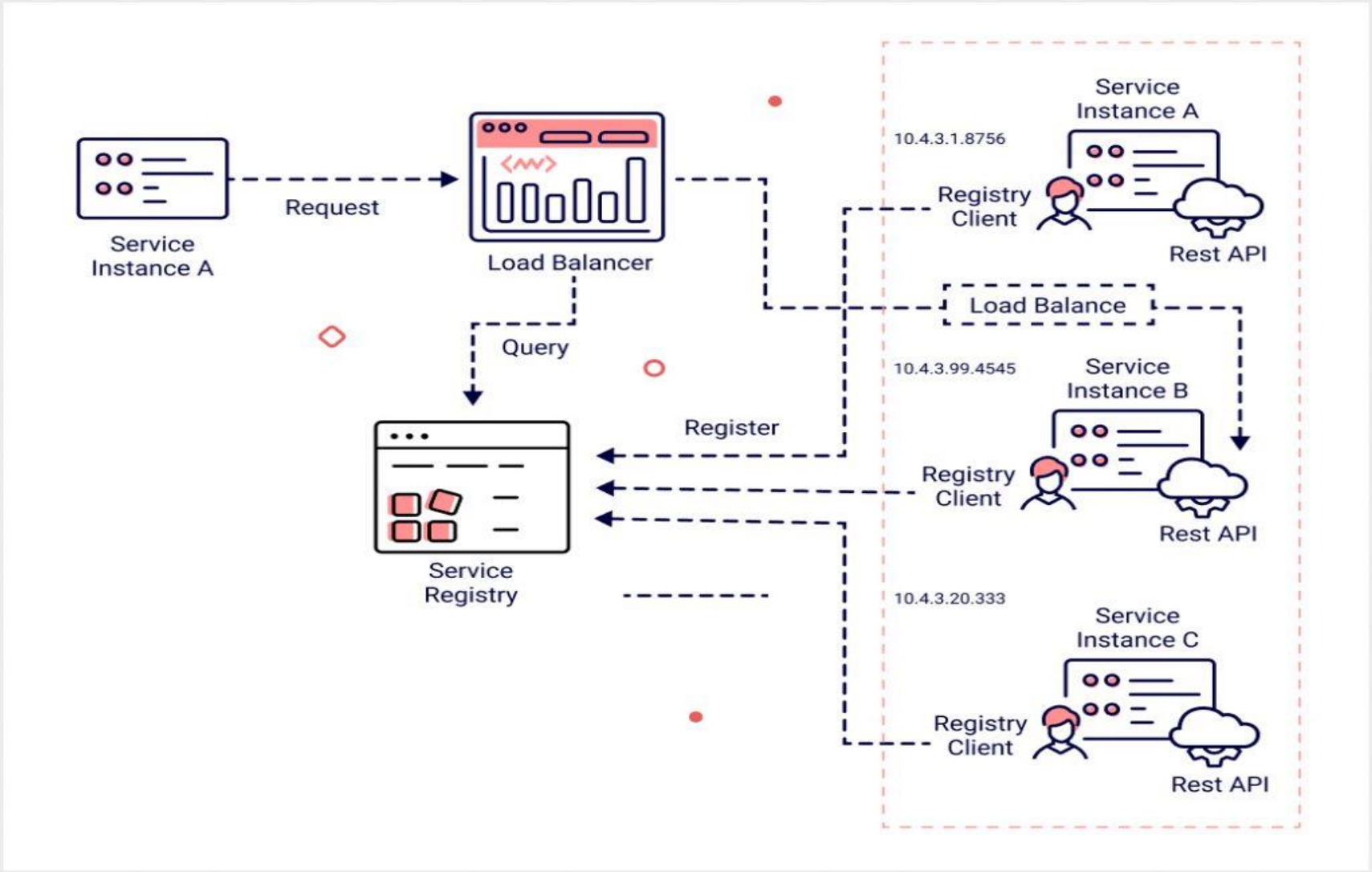
Push vs Pull

- Disadvantages:
  - Introduces an extra component (Push gateway), which adds complexity.
  - The state of the Push gateway must be managed, as it retains data until explicitly deleted.
  - Push gateway is not designed for continuous metrics (e.g., long-lived services); it's primarily intended for short-lived jobs.

Service Discovery

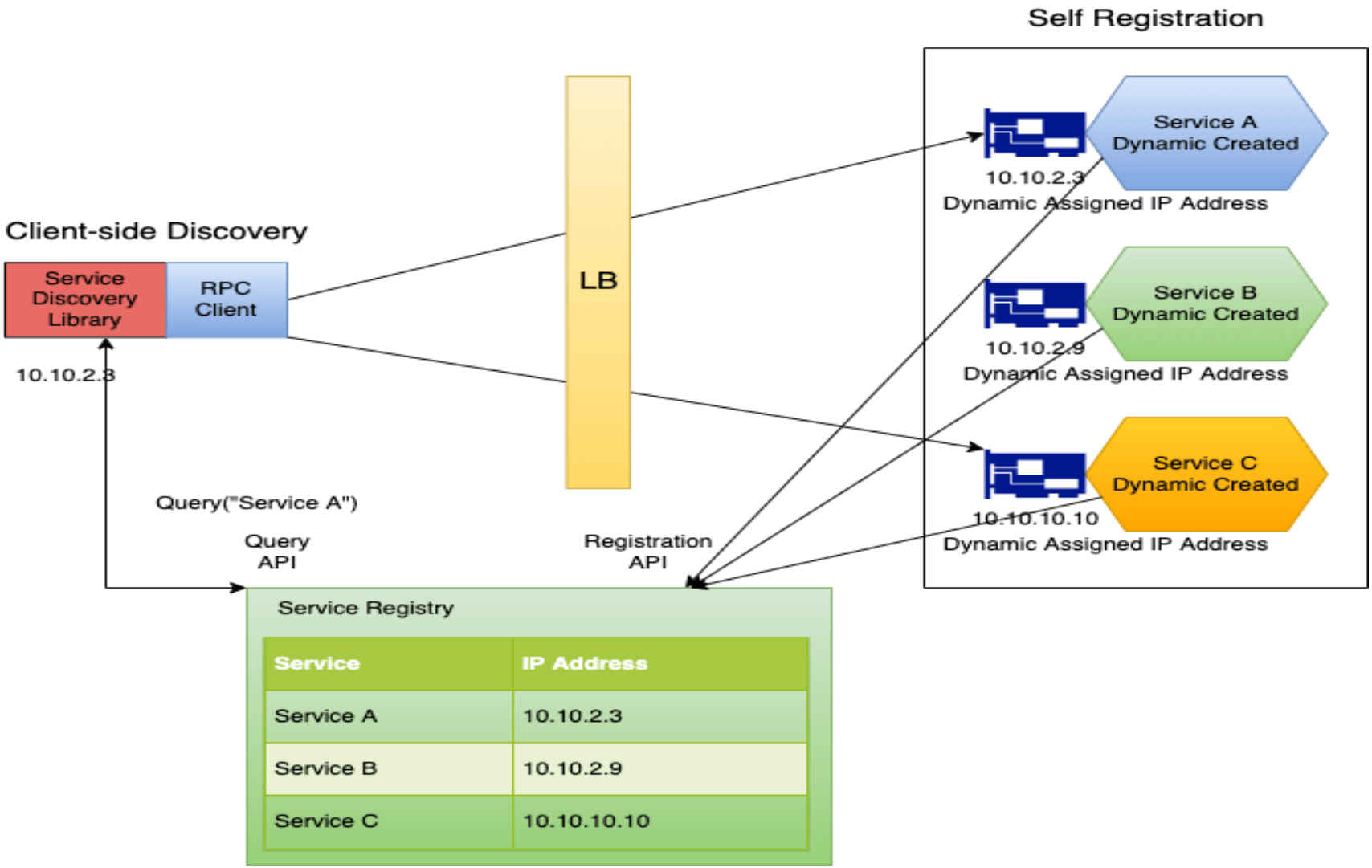
- Service discovery is a key concept in microservices and distributed systems.
- It refers to the process of automatically detecting and connecting microservices in a network.
- In these environments, services communicate with each other, and their locations (IP addresses, ports) may change dynamically due to scaling, failures, or deployments.
- Service discovery helps maintain seamless communication between services despite these changes.

Service Discovery



Service Discovery

Microservices



Service Discovery

- 1. Client-side discovery:
  - In client-side discovery, the client (the service that wants to make a request) is responsible for determining the location of service instances.
  - A service registry contains the addresses of all available services.
  - The client queries the registry to find a valid service instance and then sends a request directly to the selected instance.

Service Discovery

- **Components:**
  - **Service Registry:** A centralized database that tracks all instances of services.
  - **Client:** The service that queries the service registry and selects a service instance.
  - **Examples:**
  - Netflix Eureka (client-side service discovery)
  - Consul (can be used for client-side)

Service Discovery

- **Server-side discovery:**
  - In server-side discovery, the client sends a request to a load balancer, which in turn queries the service registry and forwards the request to an appropriate service instance.
  - The client is unaware of where the actual service instance is located; it simply sends the request to the load balancer.

Service Discovery

- **Components:**
  - **Service Registry:** Stores the locations of services.
  - **Load Balancer/Service Proxy:** Acts as an intermediary between the client and service instances.
  - **Examples:**
  - AWS Elastic Load Balancing (ELB)
  - Kubernetes (through the use of DNS or environment variables)

Service Discovery

- **Common Technologies for Service Discovery:**
  - **Eureka:** A service discovery tool developed by Netflix for client-side discovery.
  - **Consul:** A service mesh and service discovery tool from HashiCorp that supports both client-side and server-side discovery.
  - **Kubernetes:** Uses internal DNS or environment variables for service discovery within a cluster.
  - **Zookeeper:** Often used with Apache frameworks for service discovery.

Service Discovery

- **How it Works:**
  - **Service Registration:** When a new service instance starts, it registers itself with the service registry, providing details like IP address, port, and metadata.
  - **Service Deregistration:** When a service instance stops, it deregisters itself from the registry to avoid sending traffic to unavailable instances.
  - **Health Checks:** The registry may periodically perform health checks to ensure that the registered services are available and healthy.

Service Discovery

- **Benefits of Service Discovery:**
  - **Scalability:** Helps services find and communicate with each other dynamically, supporting high levels of scaling.
  - **Resilience:** Allows systems to adapt to changes in the environment, such as instances going down or new ones being added.
  - **Flexibility:** Can support different deployment environments (cloud, containers, on-premises).

SLA vs SLO vs SLI

SLA ..... ▶

SERVICE LEVEL AGREEMENT

the agreement you make with your clients or users

SLOs ..... ▶

SERVICE LEVEL OBJECTIVES

the objectives your team must hit to meet that agreement

SLIs ..... ▶

SERVICE LEVEL INDICATORS

the real numbers on your performance

Basics of SLOs, SLAs, and SLIs

- SLA (Service Level Agreement):
  - An SLA is a formal agreement between a service provider and a customer.
  - It defines the specific measurable service performance standards that the provider is obligated to meet.
  - SLAs are typically legally binding and include consequences (such as penalties or service credits) if the service provider fails to meet the agreed-upon standards.

Basics of SLOs, SLAs, and SLIs

- **Key Points:**
  - **Purpose:** Sets expectations for service performance and outlines responsibilities between the provider and customer.
  - **Enforceability:** Often legally binding with penalties for non-compliance.
  - **Example:** "The service provider guarantees 99.9% uptime over a month. If this is not met, the customer will receive a 5% refund of the monthly service fee."

Basics of SLOs, SLAs, and SLIs

- **SLO (Service Level Objective):**
  - An SLO is a specific, measurable goal that defines the acceptable performance level of a service.
  - SLOs are part of the SLA but are less formal than the SLA itself.
  - They act as internal targets for the service provider, helping to ensure that the SLAs are met.
  - The SLO is used to define performance objectives like availability, latency, or response times.

Basics of SLOs, SLAs, and SLIs

- **Key Points:**
  - **Purpose:** Sets measurable objectives that help the service provider track and maintain service levels.
  - **Measurable:** Clearly defined metrics such as uptime percentage, error rates, or latency thresholds.
  - **Example:** "The system should have an uptime of 99.9% each month."

Basics of SLOs, SLAs, and SLIs

- **SLI (Service Level Indicator):**
  - An SLI is the actual measurement or metric that indicates how well a service is performing against the SLOs.
  - SLIs are quantitative measurements, typically expressed as percentages or time units, that track specific characteristics of a service (such as availability, latency, throughput, or error rate).
  - SLIs provide the data that allows a provider to understand whether they are meeting their SLOs and SLAs.

Basics of SLOs, SLAs, and SLIs

- **Key Points:**
  - **Purpose:** Provides real-time metrics or measurements of service performance.
  - **Monitoring:** Often collected through monitoring tools and compared against SLOs.
  - **Example:** "The service achieved 99.95% uptime last month" (where uptime is the SLI).

Prometheus fundamentals

Prometheus

- Prometheus is an open-source technology designed to provide monitoring and alerting functionality for cloud-native environments, including Kubernetes.
- It can collect and store metrics as time-series data, recording information with a timestamp.
- It can also collect and record labels, which are optional key-value pairs.

Prometheus Features

- Multidimensional data model – Using time-series data, which is identified by metric name and key-value pairs.
- PromQL – A flexible querying language that can leverage the multi-dimensional data model.
- No reliance on distributed storage – All single server nodes remain autonomous.
- Pull model – Prometheus can collect time-series data by actively “pulling” data over HTTP.
- Pushing time-series data – Available through the use of an intermediary gateway.
- Monitoring target discovery – Available through static configuration or service discovery.
- Visualization – Prometheus offers multiple types of graphs and dashboards.

Micrometer Prometheus

- Micrometer provides a simple facade over the instrumentation clients for the most popular monitoring systems.
- It allows us to instrument JVM-based application code without vendor lock-in.
- Micrometer is an open-source project and provides a metric facade that exposes metric data in a vendor-neutral format that a monitoring system can understand.

Monitoring Systems

These monitoring systems are supported:

- AppOptics
- Azure Monitor
- Netflix Atlas
- CloudWatch
- Datadog
- Dynatrace
- Elastic
- Ganglia
- Graphite
- Humio
- Influx/Telegraf
- JMX
- KairosDB
- New Relic
- Prometheus
- SignalFx
- Google Stackdriver
- StatsD
- Wavefront

How Does Prometheus Monitoring Work?

- To get metrics, Prometheus requires an exposed HTTP endpoint.
- Once an endpoint is available, Prometheus can start scraping numerical data, capture it as a time series, and store it in a local database suited to time-series data.
- Prometheus can also be integrated with remote storage repositories.

How Does Prometheus Monitoring Work?

- Users can leverage queries to create temporary times series from the source.
- These series are defined by metric names and labels.
- Queries are written in PromQL, a unique language that allows users to choose and aggregate time-series data in real time.
- PromQL can also help us establish alert conditions, resulting in notifications to external systems like email, PagerDuty, or Slack.

How Does Prometheus Monitoring Work?

- Prometheus can display collected data in tabular or graph form, shown in its web-based user interface.
- We can also use APIs to integrate with third-party visualization solutions like Grafana.

What Can You Monitor with Prometheus?

- Prometheus is a versatile monitoring tool, which you can use to monitor a variety of infrastructure and application metrics.
- Use Cases
  - Service Metrics
  - Host Metrics
  - Website Uptime/Up Status
  - Cronjobs

Service Metrics

- Prometheus is typically used to collect numeric metrics from services that run 24/7.
- It allows metric data to be accessed via HTTP endpoints.
- This can be done manually or with various client libraries.
- Prometheus exposes data using a simple format, with a new line for each metric, separated with line feed characters.
- The file is published on an HTTP server that Prometheus can query and scrape metrics from based on the specified path, port, and hostname.
- Prometheus can also be used for distributed services, which are run on multiple hosts.
- Each instance publishes its own metrics and has a name that Prometheus can distinguish.

Host Metrics

- We can monitor the operating system to identify when a server's hard disk is full or if a server operates constantly at 100% CPU.
- We can install a special exporter on the host to collect the operating system information and publish it to an HTTP-reachable location.

Website Uptime/Up Status

- Prometheus doesn’t usually monitor website status, but we can use a black box exporter to enable this.
- We specify the target URL to query an endpoint.
- It performs an uptime check to receive information such as the website’s response time.
- We define the hosts to be queried in the prometheus.yml configuration file, using relabel_configs to ensure Prometheus uses the black box exporter.

Cronjobs

- To check if a cronjob is running at the specified intervals, we can use the Push Gateway to display metrics to Prometheus through an HTTP endpoint.
- We can push the timestamp of the last successful job (i.e. a backup job) to the Gateway and compare it with the current time in Prometheus.
- If the time exceeds the specified threshold, the monitor times out and triggers an alert.

Prometheus Metric Types

- The client libraries of Prometheus offer three core types of metrics.
- However, the Prometheus server does not currently save these metrics as different data types.
- Instead, it flattens all information into an untyped time series.

Prometheus Metric Types

- Counter
  - This is a cumulative metric. It represents a single monotonically-increasing counter, and its value can either increase or be reset to zero on restart.
- Gauge
  - This metric represents one numerical value, which can arbitrarily go down and up.
  - A gauge is often used to measure values like current memory usage or temperatures.
- Histogram
  - A histogram samples observations, such as request durations or response sizes.
  - It then counts the observations in a configurable bucket.
  - A histogram can also provide a total sum of all the observed values.

Prometheus Architecture

- Prometheus follows a **pull-based model** where it scrapes metrics from monitored targets at specified intervals.
- It uses a multi-dimensional data model for storing time-series data and provides a flexible query language called **PromQL** to retrieve and analyze this data.
- Prometheus is a standalone service that doesn't require external storage.

Prometheus Architecture

- **Key Components of Prometheus Architecture:**

1. **Prometheus Server:**
    1. The Prometheus server is at the core of the system architecture. It is responsible for:
        1. **Data Collection:** Scraping metrics from target services at defined intervals.
        2. **Data Storage:** Storing scraped time-series data in a local time-series database (TSDB).
        3. **Querying:** Processing queries written in PromQL to retrieve and analyze the collected data.

Prometheus Architecture

- **Targets (Instrumented Services):**
  - Targets are the systems and applications being monitored by Prometheus.
  - These targets expose metrics via an HTTP endpoint (typically /metrics), which Prometheus scrapes.
  - Prometheus interacts with these services by pulling metrics from them at regular intervals.
  - Targets can be instrumented applications, exporters, or third-party systems like databases or cloud services.

Prometheus Architecture

- **Exporters:**
- Exporters are specialized services that expose metrics from third-party systems or non-instrumented services in a Prometheus-compatible format.
- Common exporters:
  - **Node Exporter:** Exposes system-level metrics (CPU, memory, disk, etc.).
  - **Blackbox Exporter:** Probes endpoints (HTTP, DNS, ICMP) to check for uptime and reachability.
  - **Database Exporters (e.g., MySQL, PostgreSQL):** Export database-specific metrics.

Prometheus Architecture

- **Service Discovery:**
- Prometheus supports dynamic service discovery, which allows it to automatically find and scrape targets in dynamic environments (e.g., cloud, Kubernetes).
- It supports a variety of service discovery mechanisms:
  - Kubernetes
  - Consul
  - EC2 instances
  - DNS-based discovery
- Service discovery helps in automatically adapting to infrastructure changes, such as when services scale up or down.

Prometheus Architecture

- **Time-Series Database (TSDB):**
  - Prometheus stores all metrics as time-series data, identified by a unique metric name and label set.
  - The time-series data is stored locally on disk and compressed to optimize space.
  - Data is kept for a configurable retention period, after which it is deleted.

Prometheus Architecture

- **Alert manager:**
- **Alert manager** handles alerts generated by Prometheus when a rule condition is met (e.g., high CPU usage).
- Prometheus evaluates alerting rules against the data and pushes alerts to the Alertmanager.
- **Alert manager's Role:**
  - **Alert Deduplication:** Prevents repeated alerts for the same issue.
  - **Grouping:** Groups related alerts to reduce noise.
  - **Routing:** Sends alerts to the appropriate receiver (email, Slack, PagerDuty, etc.).
  - **Silencing:** Temporarily suppresses alerts during planned maintenance.

Prometheus Architecture

- **Prometheus Client Libraries:**
  - Prometheus provides client libraries for different programming languages (Go, Python, Java, Ruby, etc.) to enable custom metrics instrumentation in applications.
  - Developers can use these libraries to instrument their code and expose custom application-specific metrics.

Prometheus Architecture

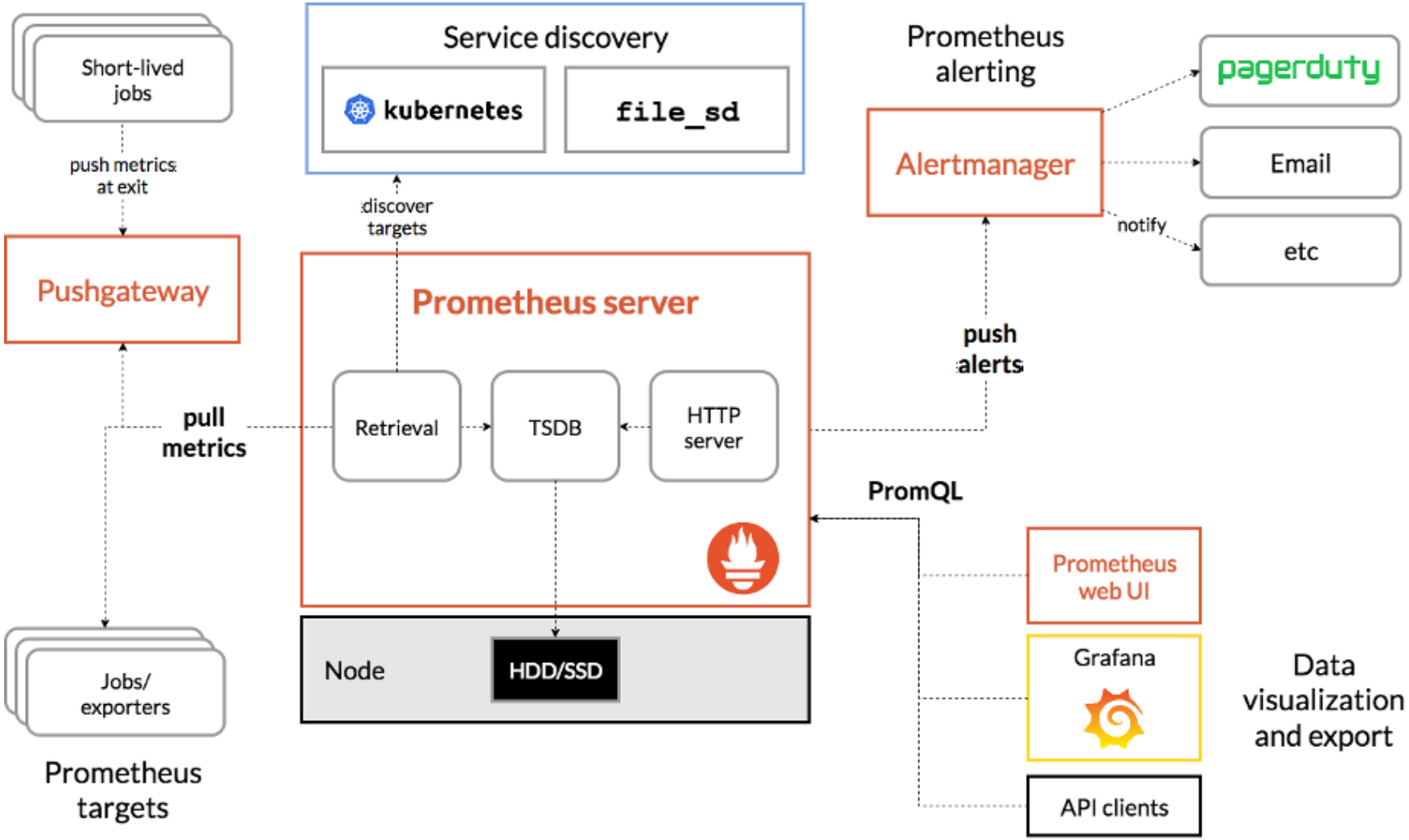
- **PromQL (Prometheus Query Language):**
  - Prometheus comes with its own powerful query language, PromQL, which allows users to select and aggregate time-series data.
  - PromQL supports complex operations like rate calculations, aggregations, and joins across metrics.

Prometheus Architecture

1. **Pushgateway (Optional):**
    1. Prometheus uses a pull model, but sometimes systems may not be suited to be scraped (e.g., batch jobs or short-lived processes).
    2. The **Pushgateway** allows these systems to push metrics to Prometheus. It acts as an intermediary for these metrics, storing them until Prometheus pulls them.

Prometheus Component Architecture

Microservices



Prometheus Server

- The Prometheus server handles the scraping and storing of metrics.
- The server manages scheduling of monitoring jobs – querying data sources (called “instances”) at a predefined polling frequency.
- Monitoring jobs are configured via one or more “scrape config” directives, managed via a YAML configuration file.
- It can be live-reloaded using a SIGHUP or the Management API.

Prometheus Server

- Prometheus relies heavily on various service discovery (SD) mechanisms to identify targets to scrape.
- These service discovery integrations range from generic interfaces like a file-based service discovery that custom SD implementations can leverage by managing a JSON or YAML file containing a list of targets.

Prometheus Server

- Prometheus also provides number of platform-specific SD implementations, including: Kubernetes, AWS EC2, Azure, GCE, Docker Swarm, OpenStack, and more.
- These generally integrate with the corresponding APIs to query the platform for targets running applications and exporters that can be scraped by Prometheus.

Prometheus Data Flow

- Scraping:
  - Prometheus scrapes metrics from the targets (applications, services, exporters) at a configured interval via HTTP. Each target exposes an endpoint that Prometheus polls for metrics.
- Storing:
  - Scraped metrics are stored in Prometheus's internal time-series database.
  - Each metric is stored as a time-series, identified by a combination of a metric name and key-value labels.
- Example metric format:

```
http_requests_total{method="GET", handler="/api"} 12893
```

Prometheus Data Flow

- Querying:
  - Users or external systems (like dashboards) can query the stored time-series data using PromQL.
  - Queries can be used to retrieve data for monitoring, debugging, or performance analysis.
  - Alerting:
  - Prometheus evaluates alerting rules against the stored data at regular intervals.
  - When an alert condition is met (e.g., CPU usage > 90% for 5 minutes), an alert is triggered and sent to the Alert manager.
  - Alert manager processes and routes the alerts to defined destinations like Slack, PagerDuty, or email.

Key Features and Benefits of Prometheus Architecture

- Pull-based Model:
  - Prometheus's pull model allows it to periodically scrape metrics from targets, making it more efficient and flexible in discovering new services or handling dynamic infrastructure.
- Time-Series Data Storage:
  - Prometheus uses a highly optimized time-series database, enabling efficient storage and querying of millions of metrics.
- Multi-dimensional Data Model:
  - Prometheus uses labels to add dimensions to metrics, making it easier to filter and group metrics based on these labels.

Key Features and Benefits of Prometheus Architecture

- PromQL:
  - The powerful query language enables deep insights into system behavior through metrics aggregation, filtering, and real-time analysis.
- Service Discovery:
  - Dynamic environments like Kubernetes benefit from Prometheus's ability to auto-discover new targets, making it a natural choice for cloud-native monitoring.
- Self-Contained:
  - Prometheus operates independently, without external dependencies, and can be scaled by federating multiple Prometheus servers.

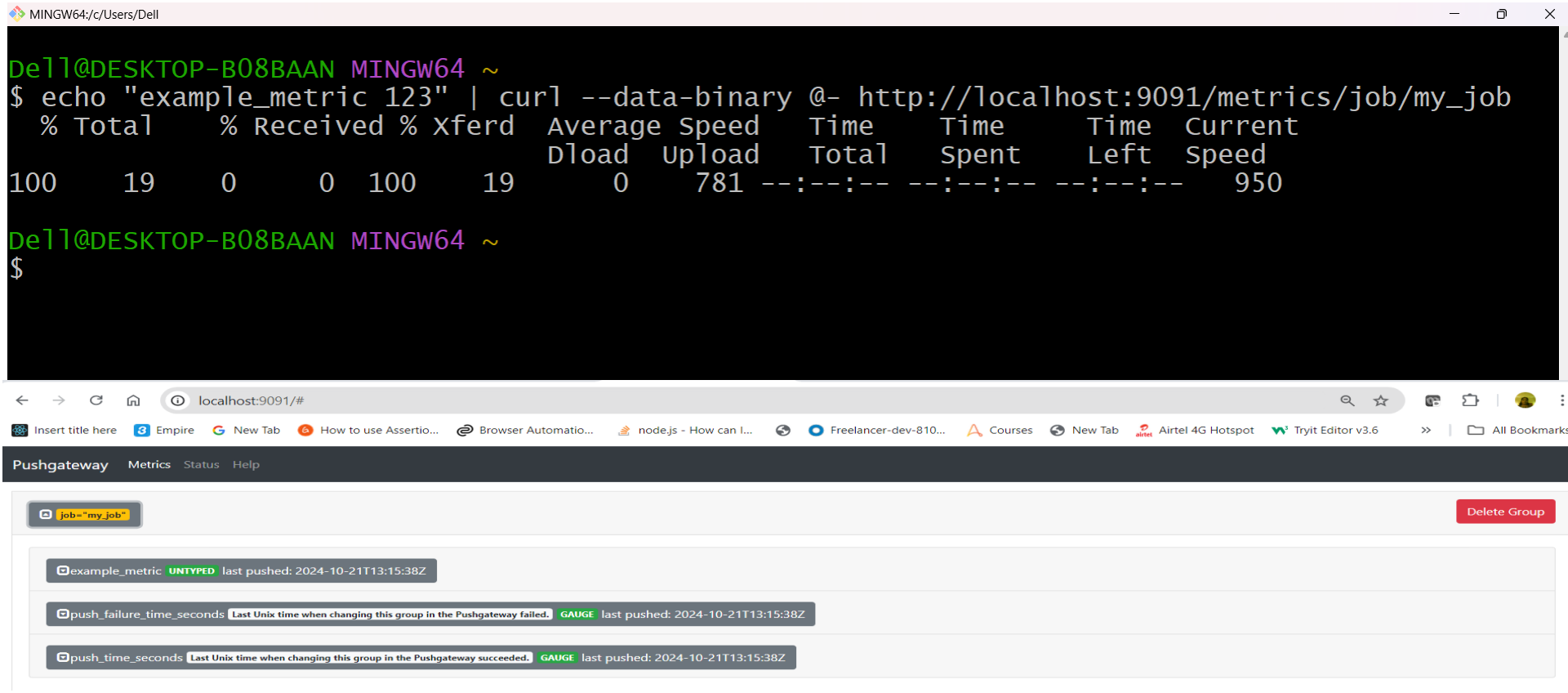
Integration Systems

- Grafana:
  - Prometheus is often integrated with Grafana for visualizing metrics using dashboards.
- Thanos/Cortex:
  - For long-term storage and scalability, Prometheus can be extended with projects like Thanos or Cortex to store metrics across multiple Prometheus instances and enable long-term storage and queries.

Push gateway send metrics - Bashscript

Microservices

- echo "example_metric 123" | curl --data-binary @- http://localhost:9091/metrics/job/my_job

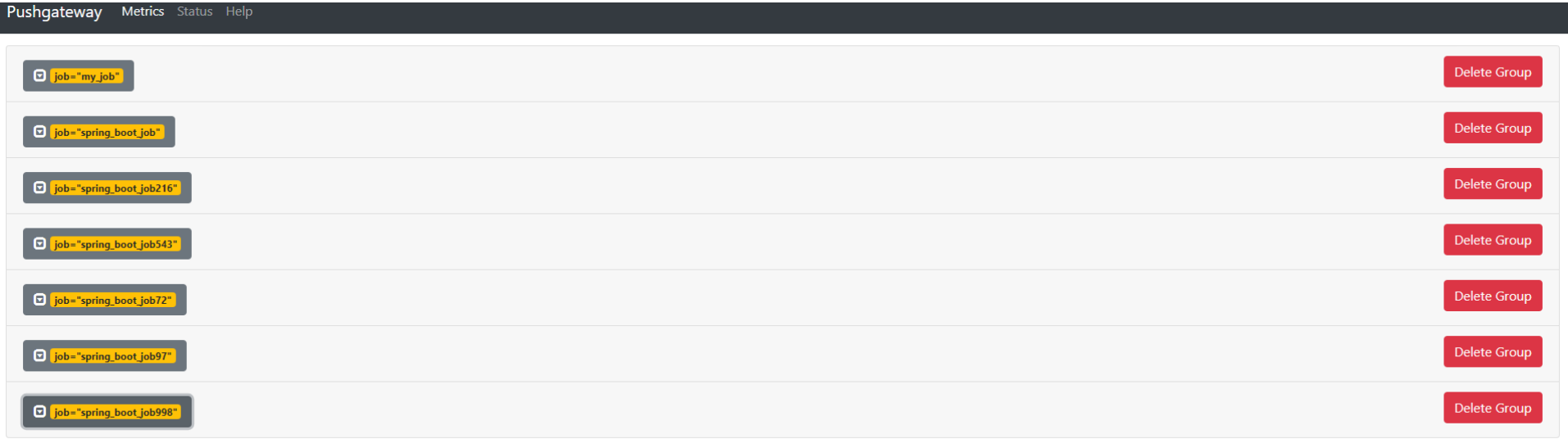


Push gateway send metrics - Bashscript

Administrator: Command Prompt

```
F:\Local disk\prometheus\PrometheusGrafana_Latest\prometheushelm\cronjobs>schtasks /create /sc
 minute /mo 1 /tn "MyCronJob" /tr "F:\Local disk\prometheus\PrometheusGrafana_Latest\prometheu
shelm\cronjobs\job.bat" /st 00:00
SUCCESS: The scheduled task "MyCronJob" has successfully been created.

F:\Local disk\prometheus\PrometheusGrafana_Latest\prometheushelm\cronjobs>
```

Push gateway send metrics – Spring boot scheduler



Push gateway send metrics – Spring boot scheduler


```java
// This method simulates a job that runs every 15 seconds
@Scheduled(cron = "0 * * * * *")
public void pushMetrics() {
    // Record the start time
    long startTime = System.currentTimeMillis();

    // Simulate a job (you can replace this with actual logic)
    performJob();

    // Record the end time and calculate the job duration
    long duration = (System.currentTimeMillis() - startTime) / 1000;
    jobDurationGauge.set(duration);  // Set the gauge with job duration

    // Labels for the job (you can add additional labels as needed)
    Map<String, String> groupingKey = new HashMap<>();
    groupingKey.put("job", "spring_boot_job"+new Random().nextInt( origin: 1, bound: 1000));

    try {
        // Push the metrics to the Prometheus Pushgateway
        pushGateway.pushAdd(registry,  job: "spring_boot_job"+new Random().nextInt( origin: 1, bound: 1000), groupingKey);
    } catch (IOException e) {
        e.printStackTrace();
    }
```

PromQL

- **PromQL (Prometheus Query Language)** is a powerful, flexible query language specifically designed for working with Prometheus's multi-dimensional time-series data.
- It enables you to extract, process, and aggregate data stored in Prometheus's time-series database.
- PromQL is essential for querying metrics, generating alerts, and building dashboards (e.g., in Grafana).
- It allows users to analyze time-series data in real time, making it valuable for monitoring system performance, troubleshooting issues, and visualizing trends.

Core Concepts in PromQL

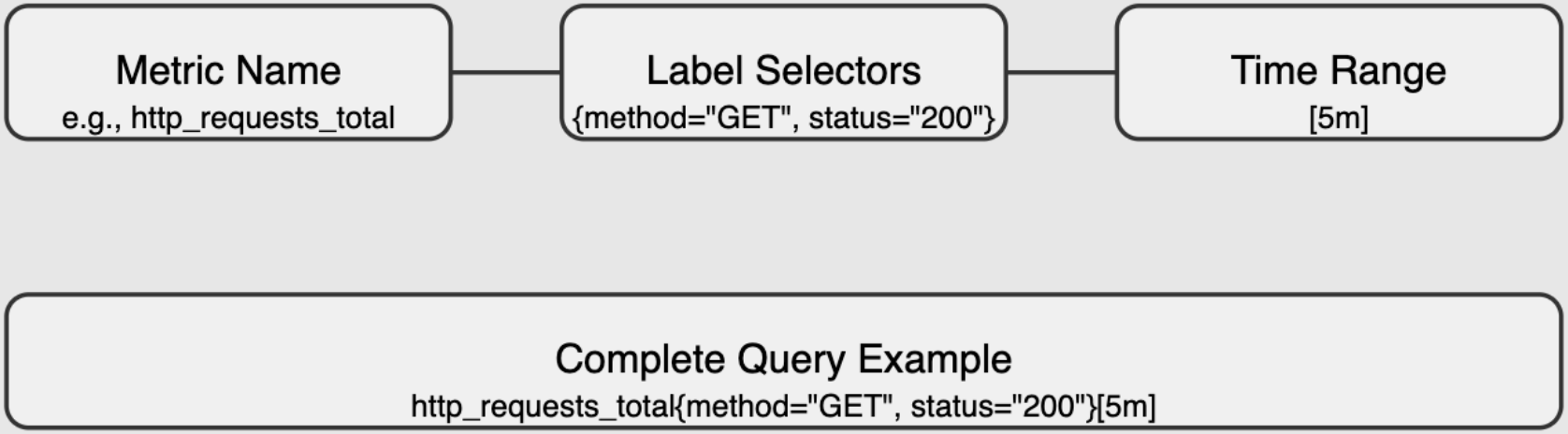
- Prometheus data is structured as time-series, which are streams of timestamped values belonging to the same metric and set of labels.
- PromQL helps extract, filter, and manipulate these time-series.
  - Metric:
    - A type of data collected over time (e.g., http_requests_total, cpu_usage).
  - Labels:
    - Key-value pairs that differentiate time-series of the same metric (e.g., {method="GET", status="200"}).
  - Instant Vector:
    - A set of time-series, each with a single value per metric at a specific timestamp.
  - Range Vector:
    - A set of time-series, each with multiple values over a time range.
  - Scalar:
    - A single numerical value (e.g., the result of a calculation).String: A textual result (e.g., up status for a target).

Core Concepts in PromQL

- **Query Structure:**
  - PromQL queries can operate on both **instant vectors** (data at a specific time) and **range vectors** (data over a time range).
  - Queries typically consist of a combination of metric names, filters, and functions.

Core Concepts in PromQL

Anatomy of a Basic PromQL Query



This query selects the 'http_requests_total' metric for GET requests with status 200 over the last 5 minutes

Selecting Data

- The very basic beginnings of any query language is being able to select data from your metrics collection.
- Metric name: Querying for all time series data collected for that metric.
- Label: Using one or more assigned labels filters metric output.
- Timestamp: Fixes a query single moment in time of your choosing.
- Range of time: Setting a query to select over a certain period of time.
- Time-shifted data: Setting a query to select over a period of time while adding an offset from the time of query execution.

Basic PromQL Queries

1. **Select All Data for a Metric**

To select the latest value of a specific metric:

```promql
metric_name
```

Copy code

For example:

```promql
http_requests_total
```

Copy code

This query retrieves the current value for the `http_requests_total` metric.

Basic PromQL Queries

2. **Select Data with Specific Labels**

You can filter results based on specific label values:

promql Copy code

```
metric_name{label1="value1", label2="value2"}
```

Example:

promql Copy code

```
http_requests_total{method="GET", handler="/api/users"}
```

This selects the total number of HTTP GET requests for the `/api/users` endpoint.

Basic PromQL Queries

3. **Select a Range of Data (Time Range)**

To query data over a specified time range:

```promql
metric_name[<duration>]
```

Copy code

Example:

```promql
http_requests_total[5m]
```

Copy code

This returns the `http_requests_total` metric over the last 5 minutes.

Basic PromQL Queries

Aggregation Queries

4. **Sum of All Values**

To sum the values of a metric across all instances or labels:

```promql
sum(metric_name)
```

Example:

```promql
sum(http_requests_total)
```

This returns the total number of HTTP requests across all instances.

Basic PromQL Queries

5. **Average of Values**

To calculate the average value across instances:

promql Copy code

```
avg(metric_name)
```

Example:

promql Copy code

```
avg(cpu_usage_seconds_total)
```

This returns the average CPU usage across all instances.

Basic PromQL Queries

6. **Max or Min of Values**

To find the maximum or minimum value across instances:

- Max:

```promql
max(metric_name)
```

- Min:

```promql
min(metric_name)
```

Example:

```promql
max(memory_usage_bytes)
```

This returns the maximum memory usage across instances.

Basic PromQL Queries

7. **Group by Label**

You can group the results by a label using the `by` clause:

promql Copy code

```
sum(metric_name) by (label)
```

Example:

promql Copy code

```
sum(http_requests_total) by (handler)
```

This returns the total number of HTTP requests grouped by the `handler` label.

Basic PromQL Queries

Rate Queries

9. **Rate of Change**

To calculate the per-second rate of change for a counter over a time range, use the `rate` function:

promql Copy code

```
rate(metric_name[<duration>])
```

Example:

promql Copy code

```
rate(http_requests_total[5m])
```

This calculates the per-second rate of HTTP requests over the last 5 minutes.

Basic PromQL Queries

10. Increase in Counter Over Time

To calculate how much a counter has increased over a time range:

```promql
increase(metric_name[<duration>])
```

Example:

```promql
increase(http_requests_total[1h])
```

Basic PromQL Queries

11. Changes Over Time (for non-counters)

To compute the difference over time for non-counter metrics (like gauges):

promql Copy code

```
delta(metric_name[<duration>])
```

Example:

promql Copy code

```
delta(cpu_usage_seconds_total[1h])
```

This calculates the change in CPU usage over the last hour.

Basic PromQL Queries

12. Histogram Quantiles

To compute percentiles (e.g., 90th percentile) for histogram metrics, use `histogram_quantile()`:

promql Copy code

```
histogram_quantile(quantile, sum(rate(histogram_metric_bucket[<duration>])) by (le))
```

Example:

promql Copy code

```
histogram_quantile(0.9, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))
```

This calculates the 90th percentile of HTTP request durations over the last 5 minutes.

Basic PromQL Queries

13. Top N

To retrieve the top N results based on a metric value:

promql Copy code

```
topk(N, metric_name)
```

Example:

promql Copy code

```
topk(5, sum(rate(http_requests_total[5m])) by (instance))
```

Basic PromQL Queries

14. Bottom N

To retrieve the bottom N results based on a metric value:

promql Copy code

```
bottomk(N, metric_name)
```

Example:

promql Copy code

```
bottomk(5, sum(rate(http_requests_total[5m])) by (instance))
```

This returns the 5 instances with the lowest request rate.

Basic PromQL Queries

15. Adding, Subtracting, Multiplying Metrics

You can apply mathematical operations to metrics.

- **Addition**:

```promql
metric_name1 + metric_name2
```

- **Subtraction**:

```promql
metric_name1 - metric_name2
```

- **Multiplication**:

```promql
metric_name1 * metric_name2
```

Basic PromQL Queries

Example:

promql

Copy code

```
cpu_usage_seconds_total * 100 / instance_cpu_quota
```

Basic PromQL Queries

Example:

promql

Copy code

```
cpu_usage_seconds_total * 100 / instance_cpu_quota
```

Basic PromQL Queries

Subquery

Return the 5-minute rate of the `http_requests_total` metric for the past 30 minutes, with a resolution of 1 minute.

```
rate(http_requests_total[5m])[30m:1m]
```

This is an example of a nested subquery. The subquery for the `deriv` function uses the default resolution. Note that using subqueries unnecessarily is unwise.

```
max_over_time(deriv(rate(distance_covered_total[5s])[30s:5s])[10m:])
```

Prometheus Query – JVM Memory Usage

- jvm_memory_used_bytes{area="heap"}
- jvm_memory_used_bytes{area="nonheap"}
- jvm_memory_max_bytes{area="heap"}
- (jvm_memory_used_bytes{area="heap"} / jvm_memory_max_bytes{area="heap"}) * 100

Prometheus Query – JVM Garbage Collection (GC)

- sum(jvm_gc_pause_seconds_sum)
- sum(jvm_gc_pause_seconds_count)
- sum(jvm_gc_pause_seconds_sum) / sum(jvm_gc_pause_seconds_count)
- sum(jvm_gc_pause_seconds_sum{gc="G1 Old Generation"})

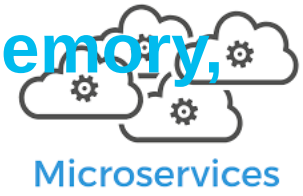
Prometheus Query – JVM Threads

- jvm_threads_live_threads
- jvm_threads_peak_threads
- jvm_threads_daemon_threads

Prometheus Query – HTTP Request Metrics

- sum(http_server_requests_seconds_count)
- sum(http_server_requests_seconds_sum) / sum(http_server_requests_seconds_count)
- sum(http_server_requests_seconds_count) by (status)

Prometheus Query – Rate of Change (GC, Memory, etc.)



- rate(jvm_gc_pause_seconds_count[5m])
- rate(jvm_memory_used_bytes{area="heap"}[5m])
- rate(http_server_requests_seconds_count[1m])

Common PromQL Query Types

PromQL also supports standard operators for combining and comparing time-series.

1. **Arithmetic Operators:**

These include basic mathematical operations such as `+`, `-`, `*`, `/`.

Example:

```promql
(rate(http_requests_total[5m]) / sum(rate(http_requests_total[5m]))) * 100
```

This calculates the percentage of total HTTP requests handled by each instance.

2. **Comparison Operators:**

Used to filter time-series based on a comparison, such as `>`, `<`, `==`, `!=`, etc.

Example:

```promql
http_requests_total > 1000
```

This returns all time-series where the number of HTTP requests is greater than 1,000.

Common PromQL Query Types

3. **Boolean Modifiers:**

These operators can also be used with a boolean modifier (e.g., `http_requests_total > 1000 == true`).

4. **Logical/Set Operators:**

These operators combine time-series based on their label sets:

- `and`: Returns time-series present in both operands.
- `or`: Returns time-series present in either operand.
- `unless`: Excludes time-series from the left operand that are also present in the right operand.

Example (and):

promql Copy code

```
cpu_usage and memory_usage
```

This returns time-series where both CPU usage and memory usage metrics are present.

Advanced Query Examples

1. **Alerting Threshold Example:**

promql Copy code

```
node_cpu_usage_seconds_total{job="node"} > 0.9
```

This query triggers an alert if CPU usage on any node exceeds 90%.

2. **Percentage of Error Rate:**

promql Copy code

```
(rate(http_requests_total{status="500"}[5m]) / rate(http_requests_total[5m])) * 100
```

This calculates the percentage of HTTP requests that returned a `500` error code.

3. **Memory Usage Over Time:**

promql Copy code

```
avg_over_time(node_memory_usage_bytes[1h])
```

This calculates the average memory usage over the last hour.

Advanced Query Examples

1. Selecting Data in PromQL

Selecting data in PromQL is the foundation of querying time-series data. You can use **instant vectors** and **range vectors** to retrieve data.

Instant Vector:

- An instant vector returns the latest value of a time-series at a given timestamp.

  Example:

  promql Copy code

  ```
  http_requests_total
  ```

  This returns the current value of `http_requests_total` for all time-series (with different labels).

Advanced Query Examples

Range Vector:

- A range vector retrieves time-series data over a specified time range.

  Example:

  promql Copy code

  ```
  http_requests_total[5m]
  ```

  This returns all time-series for the last 5 minutes.

Filtering by Labels:

You can filter data by using label selectors.

Example:

promql Copy code

```
http_requests_total{method="GET", status="200"}
```

This selects only the series where `method` is "GET" and `status` is "200".

Advanced Query Examples

2. Rates and Derivatives in PromQL

Rates and derivatives are used to calculate the change in a time-series over time, especially useful for metrics that increment continuously (like counters).

`rate()`:

- `rate()` calculates the per-second average rate of increase for a counter over a specified time window. This is useful for computing the rate at which a metric is increasing over time.

  Example:

  promql Copy code

  ```
  rate(http_requests_total[5m])
  ```

  This calculates the per-second rate of HTTP requests over the last 5 minutes.

Advanced Query Examples

`irate()`:

- `irate()` calculates the instantaneous rate of change based on the last two data points within a time range. It's more sensitive to recent changes.

  Example:

  promql Copy code

  ```
  irate(http_requests_total[5m])
  ```

`increase()`:

- `increase()` calculates the total increase in a counter over a specified time range.

  Example:

  promql Copy code

  ```
  increase(http_requests_total[1h])
  ```

  This calculates how many HTTP requests o urred in the last hour.

Advanced Query Examples

`delta()`:

- `delta()` calculates the difference between the start and end values of a time-series in a given time range, used for gauges (non-counters).

  Example:

  promql Copy code

  ```
  delta(node_memory_usage_bytes[10m])
  ```

  This calculates the difference in memory usage over the last 10 minutes.

Advanced Query Examples

3. Aggregating Over Time

PromQL allows you to aggregate time-series data over a time window.

`avg_over_time()`:

- This function calculates the average value of the time-series in a given range.

  Example:

  promql — Copy code

  ```promql
  avg_over_time(http_requests_total[1h])
  ```

  This calculates the average number of requests over the past hour.

`sum_over_time()`:

- This function calculates the sum of all values in a time range.

  Example:

  promql — Copy code

  ```promql
  sum_over_time(node_cpu_seconds_total[30m])
  ```

  This calculates the total CPU time over the last 30 minutes.

Advanced Query Examples

`min_over_time()` and `max_over_time()`:

- These functions return the minimum or maximum value over a time range.

  Example (min):

  promql Copy code

  ```
  min_over_time(node_memory_usage_bytes[1h])
  ```

  This returns the minimum memory usage in the last hour.

`count_over_time()`:

- Counts the number of data points in the specified range.

  Example:

  promql Copy code

  ```
  count_over_time(http_requests_total[10m])
  ```

Advanced Query Examples

4. Aggregating Over Dimensions

Aggregation over dimensions allows you to reduce the dimensionality of the data by applying functions (like sum, avg) across different labels.

`sum()`:

- Aggregates across time-series by summing the values across a common label set.

  Example:

  promql Copy code

  ```
  sum(rate(http_requests_total[5m])) by (method)
  ```

  This sums the rate of HTTP requests grouped by the `method` label.

`avg()`:

- Calculates the average across time-series.

  Example:

  promql Copy code

  ```
  avg(rate(cpu_usage_seconds_total[1m])) by (instance)
  ```

  This calculates the average CPU usage rate per instance.

Advanced Query Examples

`max()` and `min()`:

- These return the maximum and minimum values across time-series.

  Example (max):

```promql
max(rate(node_cpu_seconds_total[5m])) by (instance)
```

`count()`:

- Counts the number of time-series that match a given query.

  Example:

```promql
count(up) by (job)
```

  This counts the number of instances that are up, grouped by `job`.

Advanced Query Examples

5. Binary Operators

PromQL supports binary operators to combine time-series.

Arithmetic Operators:

- Operators like `+`, `-`, `*`, `/` can be used to perform arithmetic on time-series.

  Example:

  promql — Copy code

  ```
  rate(http_requests_total[5m]) / sum(rate(http_requests_total[5m])) * 100
  ```

  This calculates the percentage of total HTTP requests handled by each instance.

Comparison Operators:

- Use `>`, `<`, `>=`, `<=`, `==`, `!=` to filter time-series based on comparison.

  Example:

  promql — Copy code

  ```
  http_requests_total > 1000
  ```

  This selects time-series where the total number of HTTP requests exceeds 1000.

Advanced Query Examples

Logical/Set Operators:

- These operators (e.g., `and`, `or`, `unless`) can combine time-series sets based on their labels.

  Example (and):

  promql Copy code

  ```
  cpu_usage and memory_usage
  ```

  Example (or):

  promql Copy code

  ```
  cpu_usage or memory_usage
  ```

Advanced Query Examples

Logical/Set Operators:

- These operators (e.g., `and`, `or`, `unless`) can combine time-series sets based on their labels.

  Example (and):

  promql Copy code

  ```
  cpu_usage and memory_usage
  ```

  Example (or):

  promql Copy code

  ```
  cpu_usage or memory_usage
  ```

Advanced Query Examples

6. Histograms

Prometheus uses **histograms** to measure distributions. A histogram measures the frequency distribution of events over a range of values (e.g., request durations).

Histogram Metrics:

Prometheus stores histograms in multiple time-series for each bucket, along with a `_bucket` suffix.

Example metric:

- `request_duration_seconds_bucket{le="0.1"}`: The number of requests with a duration of 0.1 seconds or less.

`histogram_quantile()`:

- This function is used to calculate quantiles from histogram buckets. It's commonly used to calculate percentiles like the 95th percentile of request durations.

  Example:

  promql — Copy code

  ```
  histogram_quantile(0.95, sum(rate(request_duration_seconds_bucket[5m])) by (le))
  ```

  This calculates the 95th percentile of request durations over the last 5 minutes.

Advanced Query Examples

7. Timestamp Metrics

You can extract the timestamp of the most recent data point from a time-series using the `timestamp()` function.

`timestamp()`:

- Returns the timestamp (in Unix time) of the current value of a time-series.

  Example:

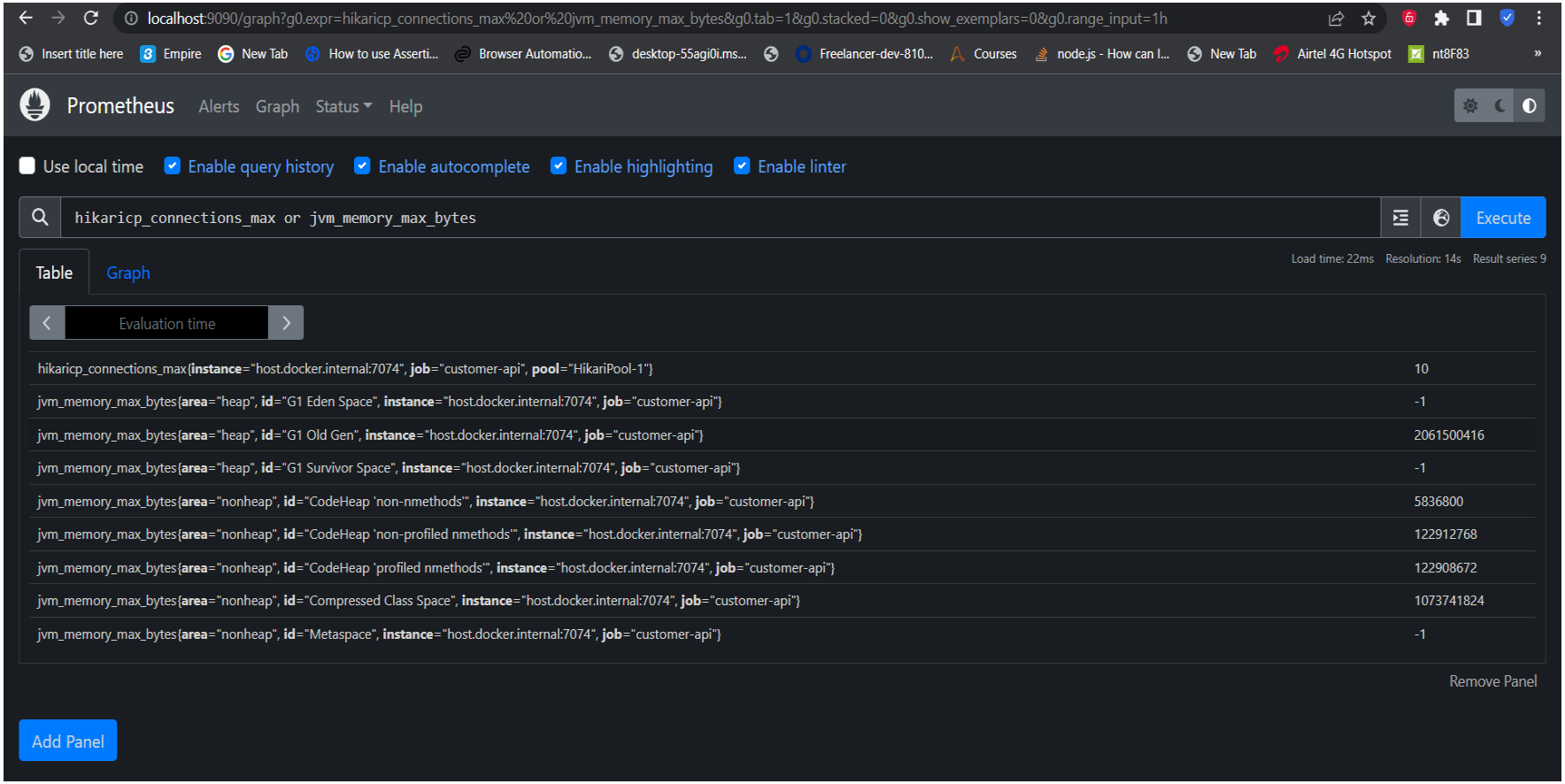
  promql Copy code

  ```
  timestamp(http_requests_total)
  ```

  This returns the Unix timestamp of the last `http_requests_total` value.

Prometheus Query

Microservices



Prometheus Exporters

- Prometheus exporters are components or services that translate metrics from various sources into a format that Prometheus can scrape and understand.
- Since Prometheus natively supports scraping metrics from its instrumentation clients, exporters are used to collect metrics from systems that do not natively support Prometheus, such as databases, hardware devices, or external applications.

Prometheus Exporters

- Applications are instrumented using client libraries.
- It enables an HTTP endpoint where internal metrics are exposed and collected by Prometheus servers.
- Other monitoring tools relies on agents or embedded instrumentation (e.g. APM client libraries) to collect data and "push" metrics to the monitoring backend.

Types of Prometheus Exporters

- Node Exporter:
  - Collects hardware and OS-level metrics like CPU usage, memory, disk I/O, and network stats from Linux or Unix-based systems.
  - Commonly used to monitor servers.
- Blackbox Exporter:
  - Allows probing of endpoints over various network protocols like HTTP, HTTPS, DNS, TCP, and ICMP.Great for uptime monitoring and checking service availability.

Types of Prometheus Exporters

- Database Exporters:
  - MySQL Exporter:
    - Exposes MySQL server metrics, such as query statistics, table locks, and replication status.
  - PostgreSQL Exporter:
    - Similar for PostgreSQL databases, providing connection, query, and performance metrics.
  - MongoDB Exporter, Redis Exporter, Elasticsearch Exporter, etc., work similarly for other databases.
- JMX Exporter:
  - Collects metrics from Java applications running with Java Management Extensions (JMX), typically for applications like Apache Kafka, Tomcat, and Cassandra.

Types of Prometheus Exporters

- HAProxy Exporter:
  - Used to gather metrics from HAProxy instances, like requests, sessions, and backend stats.
- Nginx Exporter:
  - Exposes metrics from the Nginx web server, such as request rates, response times, and error counts.
- cAdvisor Exporter:
  - Collects metrics from running containers and exposes resource usage statistics like CPU, memory, and network utilization.
- SNMP Exporter:
  - Collects metrics from devices via the Simple Network Management Protocol (SNMP), often used for monitoring network devices like switches, routers, and printers.

How Prometheus Exporters Work

- Data Collection:
  - Exporters collect metrics from an underlying source (like a database or hardware).
- Metrics Exposure:
  - Exporters transform the collected data into the Prometheus metrics format and expose it at an HTTP endpoint.
- Prometheus Scraping:
  - Prometheus scrapes these endpoints at defined intervals and stores the metrics for querying and alerting.

Example: The Prometheus Node Exporter

- It is one of the most popular exporters in the Prometheus ecosystem.
- It is used to expose hardware and operating system metrics, particularly from Linux/Unix systems.
- These metrics are essential for monitoring the health and performance of a machine or virtual server.

Example: The Prometheus Node Exporter

- **Features of Node Exporter:**
- **Wide Range of Metrics**: Collects various hardware and OS-related metrics, such as CPU usage, memory usage, disk I/O, network stats, filesystem information, and more.
- **Modular Design**: Supports many collectors, allowing for enabling or disabling specific metric groups based on your monitoring needs.
- **Minimal Resource Usage**: The exporter is designed to be lightweight, so it has minimal performance impact on the host system.

Example: The Prometheus Node Exporter

- **Common Metrics**
  - Exposed by Node Exporter:
    - CPU:node_cpu_seconds_total:
      - Total CPU time spent in each mode (user, system, idle, etc.).
  - Memory:
    - node_memory_MemAvailable_bytes: Available memory.
    - node_memory_MemFree_bytes: Free memory.
    - node_memory_SwapFree_bytes: Free swap memory.

Example: The Prometheus Node Exporter

- **Disk I/O:**
  - **node_disk_io_time_seconds_total: Time spent doing I/O operations.**
  - **node_disk_read_bytes_total: Bytes read from disk.**
  - **node_disk_written_bytes_total: Bytes written to disk.**
- **Filesystem:**
  - **node_filesystem_size_bytes: Total size of a filesystem.**
  - **node_filesystem_free_bytes: Free space available.**
  - **node_filesystem_files: Total number of inodes.**

Example: The Prometheus Node Exporter

- **Network:**
  - **node_network_receive_bytes_total: Total bytes received on a network interface.**
  - **node_network_transmit_bytes_total: Total bytes transmitted on a network interface.**
- **System Load:**
  - **node_load1: Load average over the last 1 minute.**
  - **node_load5: Load average over the last 5 minutes.**

Example: The Prometheus Node Exporter

- **scrape_configs:**
- ** - job_name: 'node_exporter'**
- **   static_configs:**
- **   - targets: ['<node-exporter-host>:9100']**

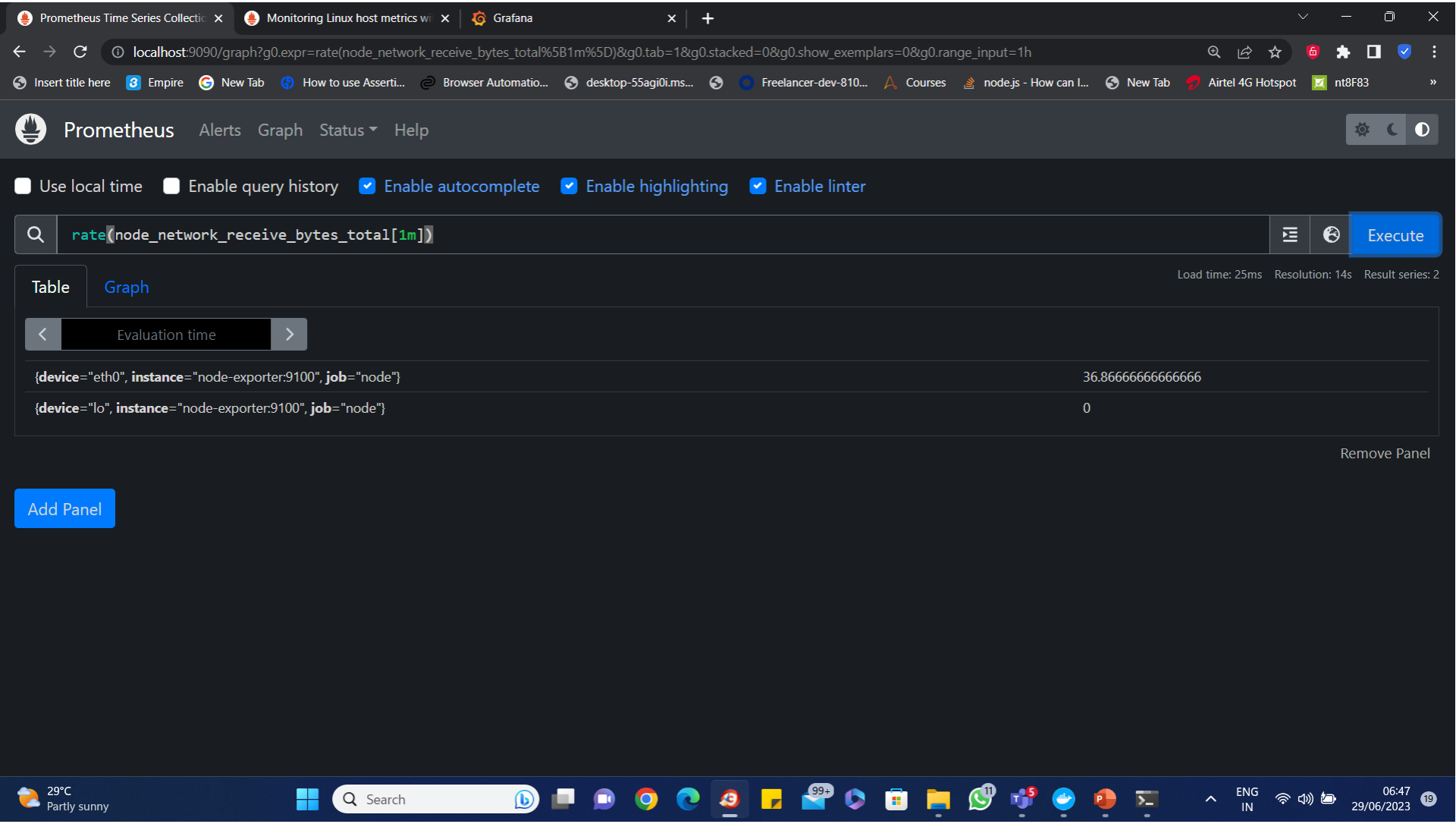
Node Exporter

- docker run -d -p 9100:9100 --net="boanetwork" prom/node-exporter

Metric	Meaning
rate(node_cpu_seconds_total{mode="system"}[1m])	The average amount of CPU time spent in system mode, per second, over the last minute (in seconds)
node_filesystem_avail_bytes	The filesystem space available to non-root users (in bytes)
rate(node_network_receive_bytes_total[1m])	The average network traffic received, per second, over the last minute (in bytes)

Node Exporter

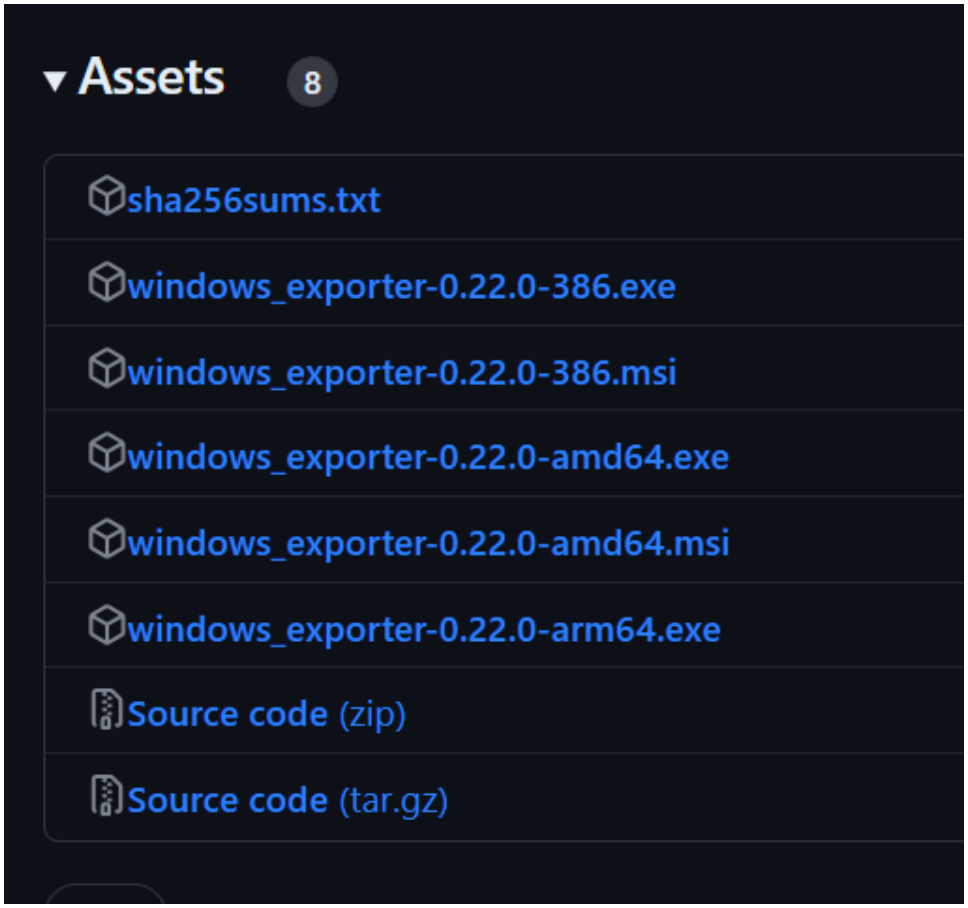
Microservices



Windows Exporter

Microservices

- https://github.com/prometheus-community/windows_exporter/releases



Windows Exporter

- http://localhost:9182/metrics

Custom Exporter


```
G:\Local disk\prometheus\customexporter>python metricexporter.py -f "/test" -e "txt"
2023-06-29 22:20:09,991 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:20:19,992 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:20:29,993 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:20:39,994 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:20:49,995 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:20:59,996 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:21:09,997 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:21:19,998 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:21:29,999 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:21:40,000 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:21:50,001 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:22:00,002 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:22:10,003 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:22:20,004 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:22:30,005 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:22:40,006 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:22:50,007 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:23:00,008 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:23:10,009 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:23:20,010 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:23:30,010 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:23:40,011 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:23:50,012 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:24:00,013 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:24:10,014 - root - INFO - Founds 2 file(s) in folder /test with extension txt
2023-06-29 22:24:20,015 - root - INFO - Founds 2 file(s) in folder /test with extension txt
```

Custom Exporter

localhost:9877/metrics

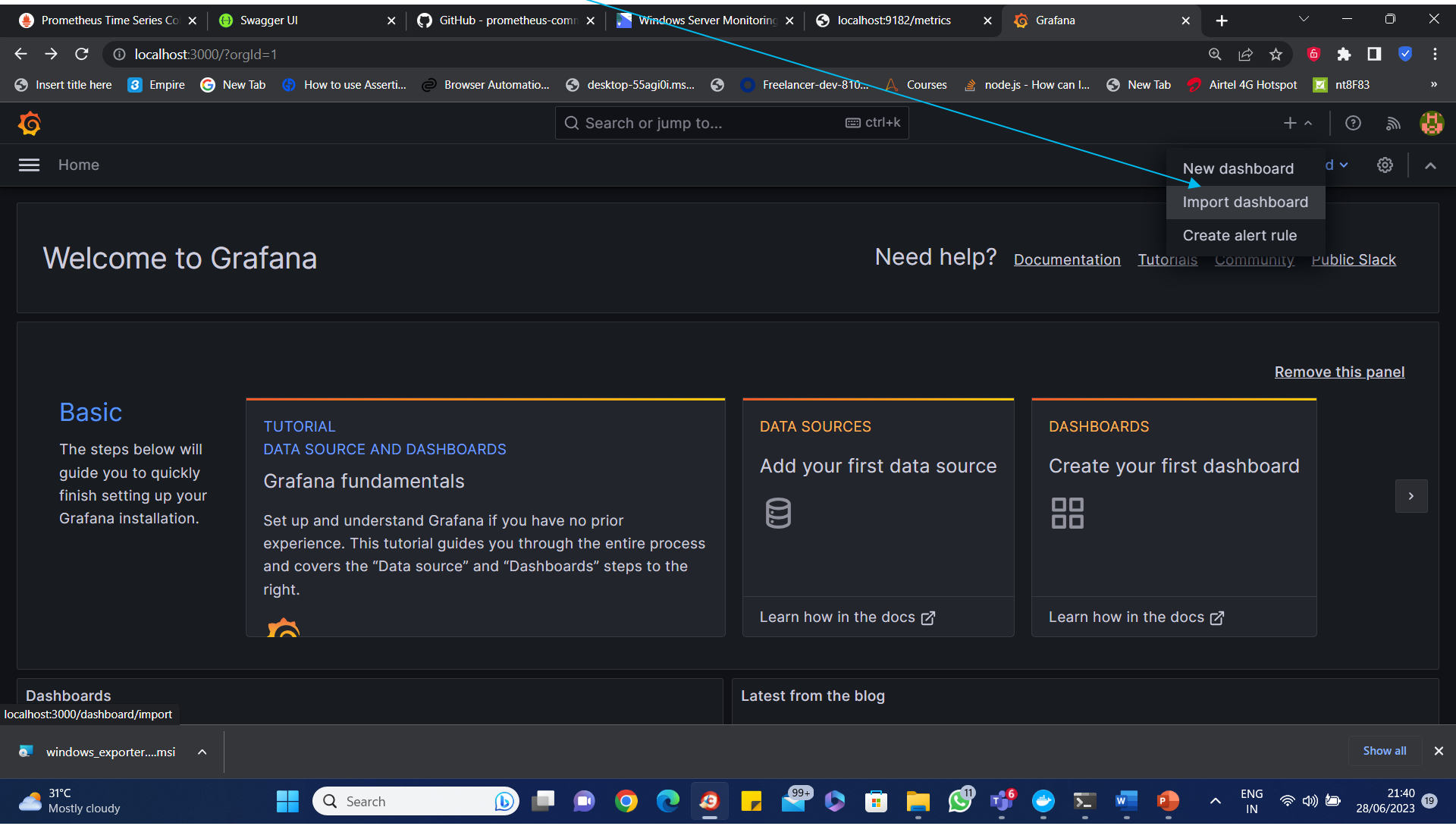
Insert title here | Empire | New Tab | How to use Asserti... | Browser Automatio... | desktop-55agi0i.ms...

Browser Automation and Acceptance Testin
https://blog.codecentric.de/en/2013/02/br

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 209.0
python_gc_objects_collected_total{generation="1"} 356.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 42.0
python_gc_collections_total{generation="1"} 3.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="11",patchlevel="4",version="3.11.4"} 1.0
# HELP cust_txt_files_in__test_total number of *txt files in /test
# TYPE cust_txt_files_in__test_total gauge
cust_txt_files_in__test_total 2.0
```

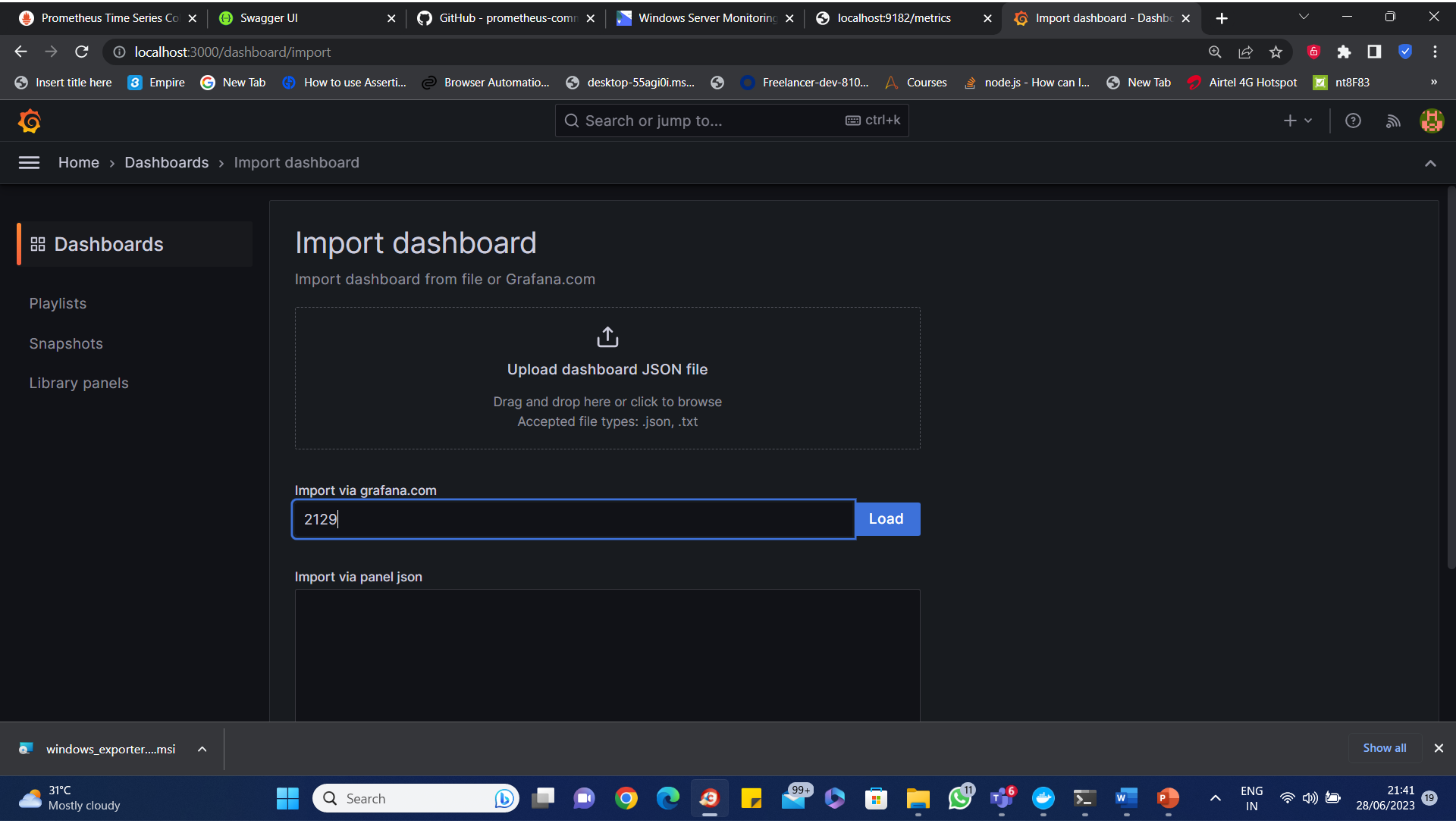
Grafana

Microservices



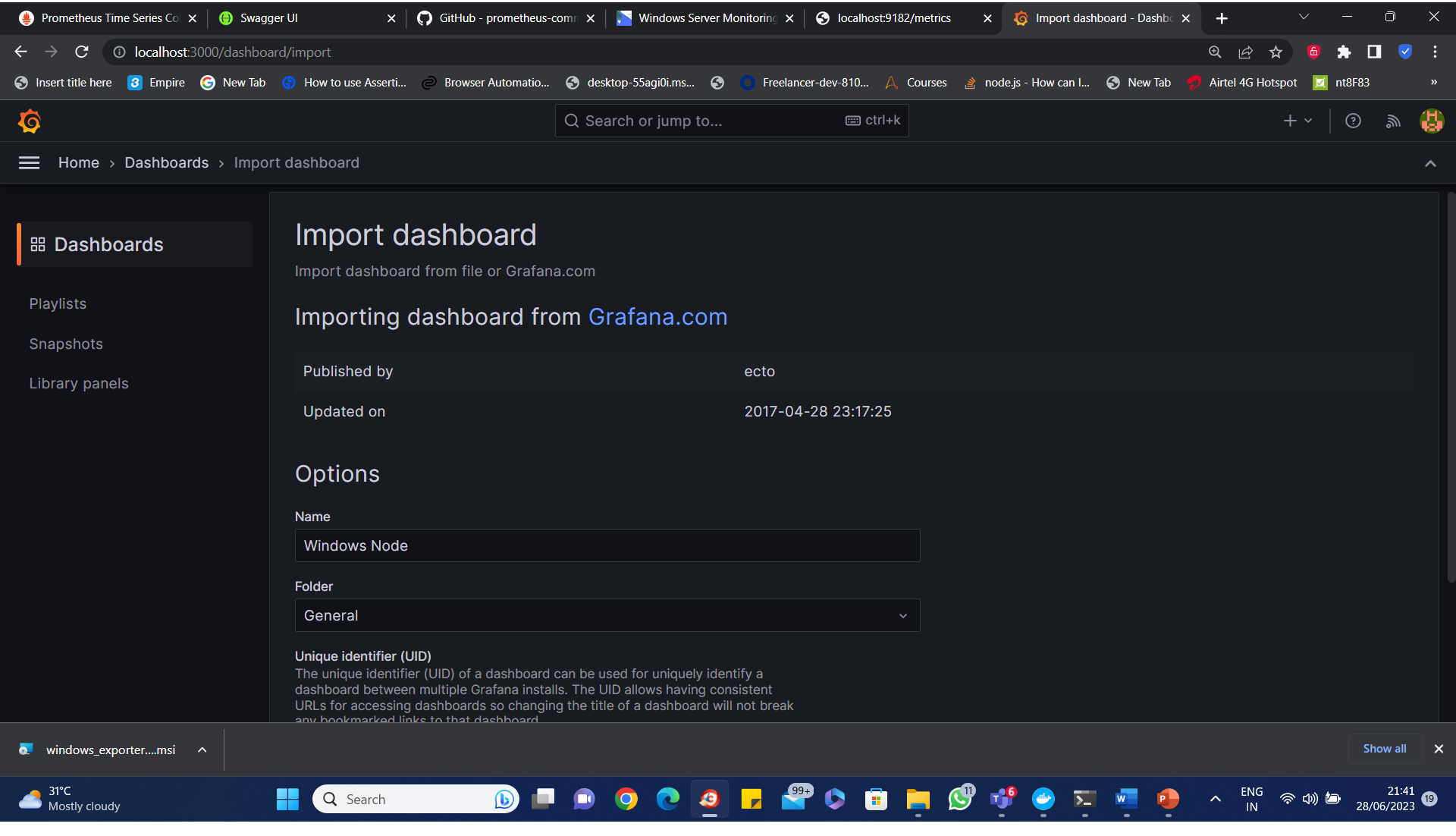
Grafana

Microservices



Grafana

Microservices



Example: The Prometheus Node Exporter

- The Prometheus node_exporter is officially distributed as a binary archive.
- Like other exporters, the node_exporter must be configured to listen on a dedicated port (9100 by default).
- For example, if a host is running a MySQL database, the official Prometheus MySQL exporter would expose metrics on port 9104 by default.
- Node_exporter would expose machine metrics on port 9100.
- Caution is advised to avoid prolonged runtimes (as measured by the built-in scrape_duration_seconds metric).

Statsd-Exporter

- The statsd-exporter is used to convert Statsd metrics into Prometheus metrics.
- This is because the StatsD text-format represents metrics differently to Prometheus' exposition format which is represented like this:
- <time series name>{<label name>=<label value>,...}

Statsd-Exporter

Prometheus statsd-exporter mapping (YAML)

```
- match: "test_api_call.*.timer.*"
  name: "test_api_call"
  labels:
    api_name:     "$1"
    api_endpoint: "$2"
```

StatsD metric

```
test_api_call.orders-api.timer./v1/orders:80|ms
```

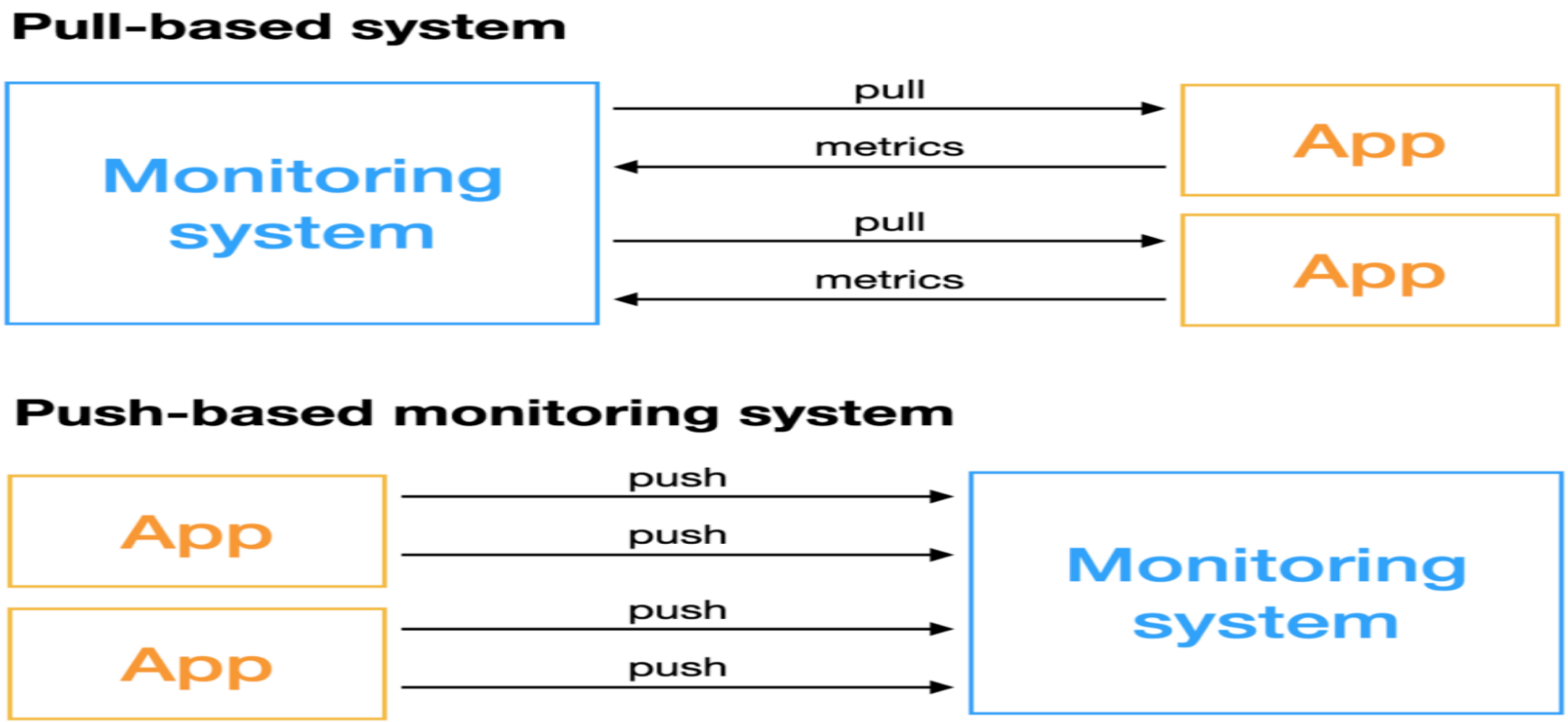
Prometheus metrics (summary)

```
test_api_call{api_name="orders-api",api_endpoint="/v1/orders",quantile="0.5"} 0.08
test_api_call{api_name="orders-api",api_endpoint="/v1/orders",quantile="0.9"} 0.08
test_api_call{api_name="orders-api",api_endpoint="/v1/orders",quantile="0.99"} 0.08
test_api_call_sum{api_name="orders-api",api_endpoint="/v1/orders"} 0.799999999999999
test_api_call_count{api_name="orders-api",api_endpoint="/v1/orders"} 10
```

time series name

labels

Types of Monitoring Systems



Types of Monitoring Systems

- The Pull-based monitoring system actively obtains indicators, and the objects.
- Monitoring system should have the capability to be accessed remotely.
- Push-based monitoring systems do not actively obtain data, but the monitored objects actively push indicators.
- There are differences between the two methods in many aspects.
- For the construction and selection of monitoring systems, we must understand the advantages and disadvantages of these two methods in advance and choose the appropriate scheme for implementation.

Types of Monitoring Systems

Feature	Details	Pull Monitoring	Push Monitoring
Principle and Deployment	Configuration	Native centralized configuration	End configuration, support centralization through configuration center
	Monitoring object discovery	Depending on service discovery mechanisms, such as Zookeeper, Etcd, and Consul registries	Reported by applications and agents without service discovery module
	Deployment method	1. Application exposes port and access service discovery, with native support for Pull protocol	1. Unified proxy of Agent, capture data from host, MySQL, and other middleware and push to the monitoring system; Agent can also
		2. Hosts, MySQL, NGINX, and other middleware rely on adapters (also known as exporters) to capture metrics and then provide Pull ports	2. Applications proactively push to the monitoring system
Scalability	Scalability	Depending on Pull-end for extension; Pull Agent needs to be decoupled from storage (native Prometheus does not support); Push Agent is divided according to shards	Simple, agent can be scaled horizontally

Types of Monitoring Systems

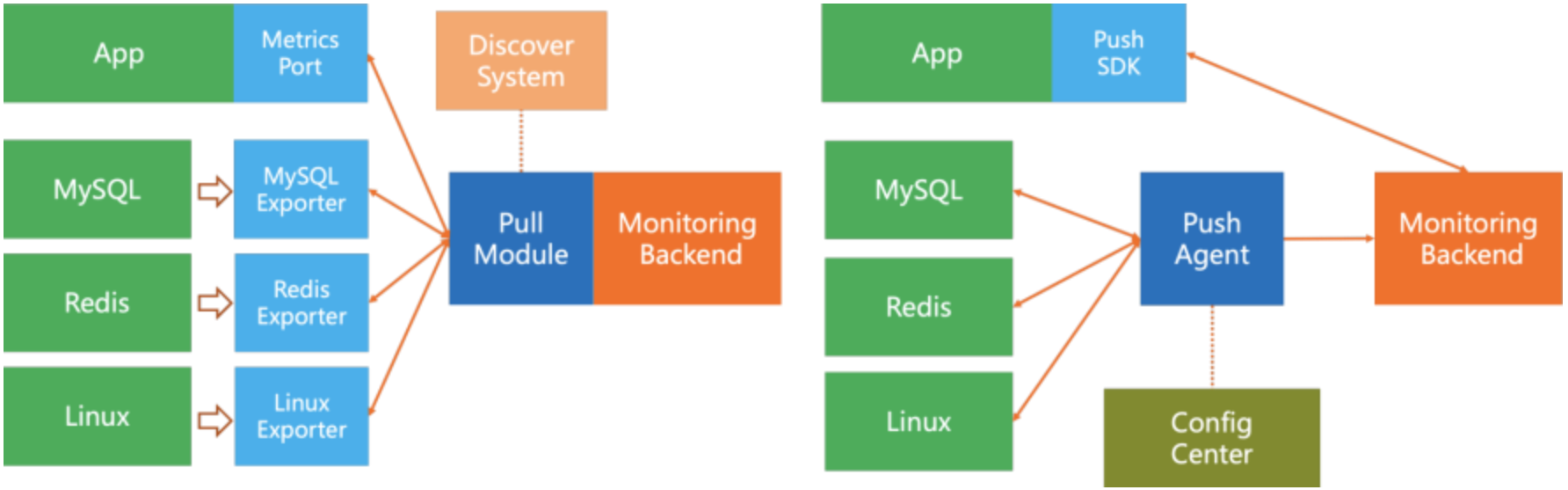
Capabilities Comparison	Monitoring object survivability	Easy	Unable to find the reason behind why the object is not alive
	Data Completeness Calculation	1. Easy when deploying at the pull-end and storage coupling-end	Difficult
		2. Difficult in the distributed deployment of Pull Agent	
	Short lifecycle (Job, Serverless)/Real-time data acquisition	Difficult to apply	Applicable
	Metric acquisition flexibility	On-demand acquisition	Passive acceptance, requires some filters for additional support
	Application coupling	Applications are decoupled from the monitoring system. Applications do not need to care about the peer address of Push and handling Push errors	Higher coupling than Pull
	Resource consumption	1. Low resource consumption in the application exposing port mode	1. Low resource consumption in application push mode
		2. High resource consumption in Exporter mode	2. Low resource consumption in Agent mode (multiple systems can be collected at the same time)
	Security assurance	The workload is heavy. It is necessary to ensure the security of the exposed port of the application and the security of the exporter port, which are vulnerable to DDoS	Low, agents and servers generally perform data transmission with encryption and authentication

Types of Monitoring Systems

Machine and Labor Cost	Core O&M consumption	1. Pull Agent stability and scale-out	1. Push Agent stability
		2. Server stability and scale-out	2. Server stability and scale-out
		3. Service discovery system stability	3. Configuration center stability and scale-out (optional)
		4. Exporter stability and scale-out	4. Network connectivity assurance (forward connectivity, relatively simple)
		5. Network connectivity assurance (reverse connectivity, cross-cluster, and network ACL)	

Types of Monitoring Systems

Pull Model Architecture



Types of Monitoring Systems

- Pull Architecture
- Service discovery system, including host service discovery (generally depends on the company's CMDB system), application service discovery (such as Consul), and PaaS service discovery (such as Kubernetes).
- The pull module needs to have the capability to connect to these service discovery systems.
- In addition to the service discovery part, the Pull core module generally uses the common protocol to pull data from the remote end.
- It usually supports configuration pull interval, timeout interval, metric filtering, rename, and simple process capabilities.
- The application-end SDK supports listening to a fixed port to provide the capability to be pulled.
- Since various middleware and other systems are incompatible with the Pull protocol, it is necessary to develop an Agent corresponding to Exporter to support pulling the metrics of these systems and provide standard Pull interfaces.

Types of Monitoring Systems

- Push Model Architecture
  - The Push model is simple and can be described as follows.
  - Push Agent supports pulling the metric data of various monitored objects and pushing them to the server.
  - It can be deployed in a manner coupled with the monitoring system or deployed separately.
  - ConfigCenter (optional) can provide centralized dynamic configuration capabilities, such as monitoring targets, collection intervals, metric filtering, metric processing, and remote targets.
  - The application-end SDK supports sending data to the monitoring backend or the local agent (usually, the local agent also implements a set of backend interfaces).

Prometheus Push gateway

- Prometheus is a primarily pull-based monitoring system.
- An additional component called the "Push gateway" is available for pushing metrics from external applications and services.
- The Push gateway is useful for collecting metrics from systems that are not compatible with the otherwise pull-based infrastructure.
- For instance, short-lived batch jobs that are ephemeral in nature may start and end before Prometheus can discover and scrape metrics from them.
- The Prometheus Pushgateway can be used to push the metrics of such processes to prevent losing essential data before they get a chance to get scraped.

Prometheus Client Libraries

- Applications only generate metrics after we add instrumentation to their code.
- We can do this via one of the Prometheus client libraries.
- Prometheus official client libraries are compatible with Java/JVM, Go, Python, and Ruby.
- Third-party client libraries are also available for Node.js, Haskell, C#/.Net, Common Lisp, Dart, Erlang, Rust, and more.
- Choose a Prometheus client library that matches the language of our application.

Prometheus Alert manager

- The Alert manager handles alerts sent by client applications such as the Prometheus server.
- In practice, the Prometheus server generates alerts when an alert condition is met in a user-defined alerting rule.
- Alert manager then manages those alerts, sending notifications via one of eight built-in notification receivers: email, PagerDuty, PushOver, Slack, OpsGenie, VictorOps, WeChat, and Webhooks.
- Once configured, Alertmanager instances handle deduplicating, grouping, inhibition, silencing, and routing of alerts to the correct receiver.

Prometheus Alert manager

- Prometheus alerting is separated into two parts (alerting rules and alert processing).
- Configuring Alert manager is only half of the battle.
- The bulk of the business logic around alerting in Prometheus is based on alerting rules.
- It can be configured in Prometheus itself.
- Alerting rules are PromQL expressions that are evaluated at regular intervals by the Prometheus server.
- By default, all Prometheus rules (recording rules and alerting rules) are evaluated on a 1m interval, but this is configurable (see the evaluation_interval parameter).

Prometheus Alert manager

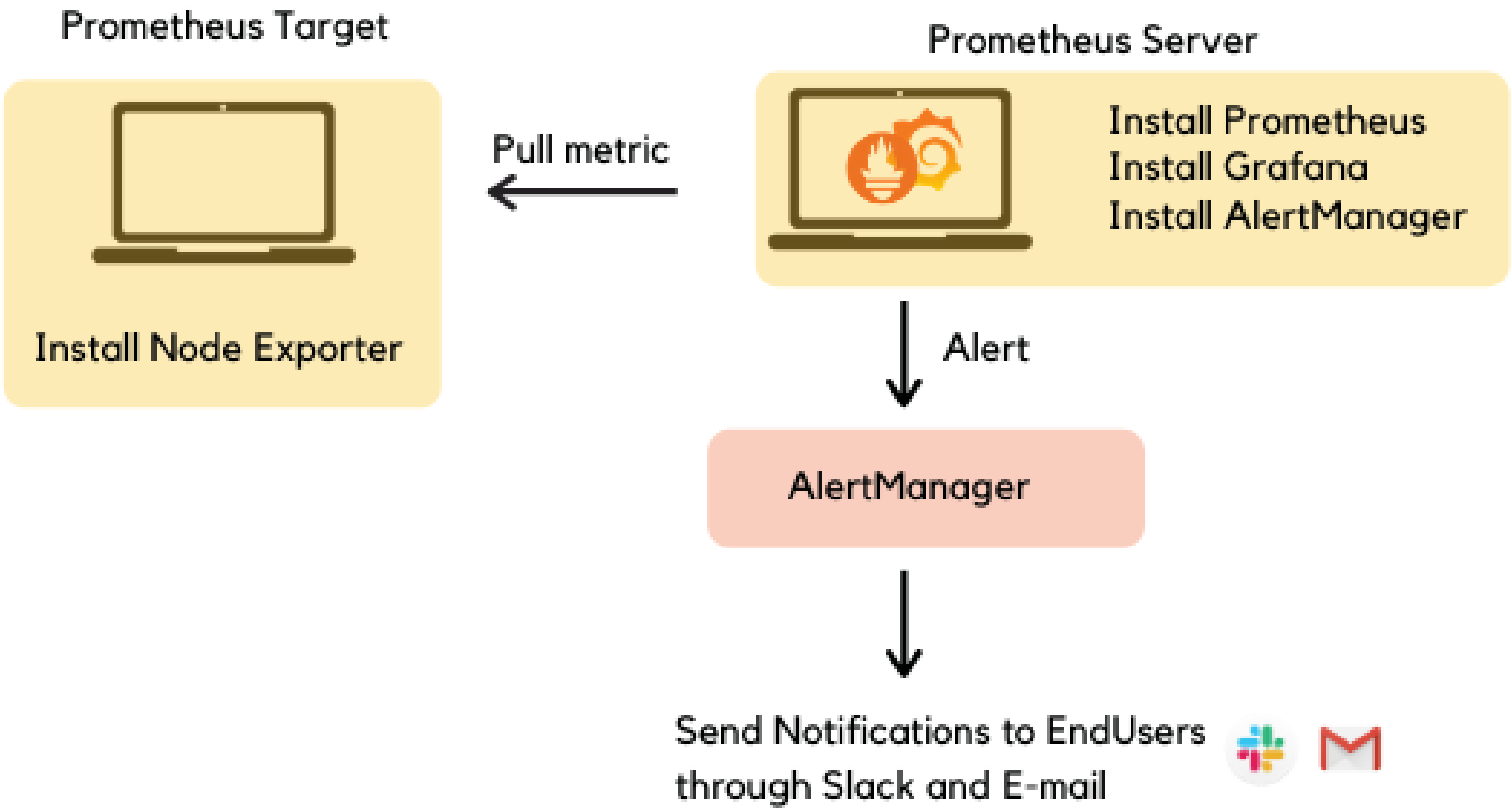
- Alert Manager is a single binary which handles alerts sent by Prometheus server and notifies end user through E-mail, Slack or other tools.
- Monitoring helps predict potential problem or notify for current problem in our system and gives detail regarding the problem.
- Alerting helps notify as soon as the problem occurs and allow teams to identify problem through notifications.

Prometheus Alert manager

- Alerting with Prometheus setup steps are mentioned below:
  - Setup and configure Alert Manager.
  - Configure the config file on Prometheus so it can talk to the Alert Manager.
  - Define alert rules in Prometheus server configuration.
  - Define alert mechanism in Alert Manager to send alerts via Slack and Mail.

Prometheus Alert manager

Microservices



Prometheus Alert manager

- Alert Manager which manages the alerts through its pipeline of silencing, inhibition, grouping and sending out notifications.
- Silencing is to mute alerts for a given time.
- Alerts are checked to match against active silent alerts, if a match is found then no notifications are sent.
- Inhibition is to suppress notifications for certain alerts if other alerts are already fired.
- Grouping group alerts of similar nature into a single notification.
- This helps prevent firing multiple notifications simultaneously to the receivers like Mail or Slack.

Configuring Prometheus to monitor itself

- Prometheus collects metrics from targets by scraping metrics HTTP endpoints.
- Since Prometheus exposes data to itself, it can also scrape and monitor its own health.

Configuring Prometheus to monitor itself


```
global:
  scrape_interval:     15s # By default, scrape targets every 15 seconds.

  # Attach these labels to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
    monitor: 'codelab-monitor'

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s

    static_configs:
      - targets: ['localhost:9090']
```

Start the Prometheus

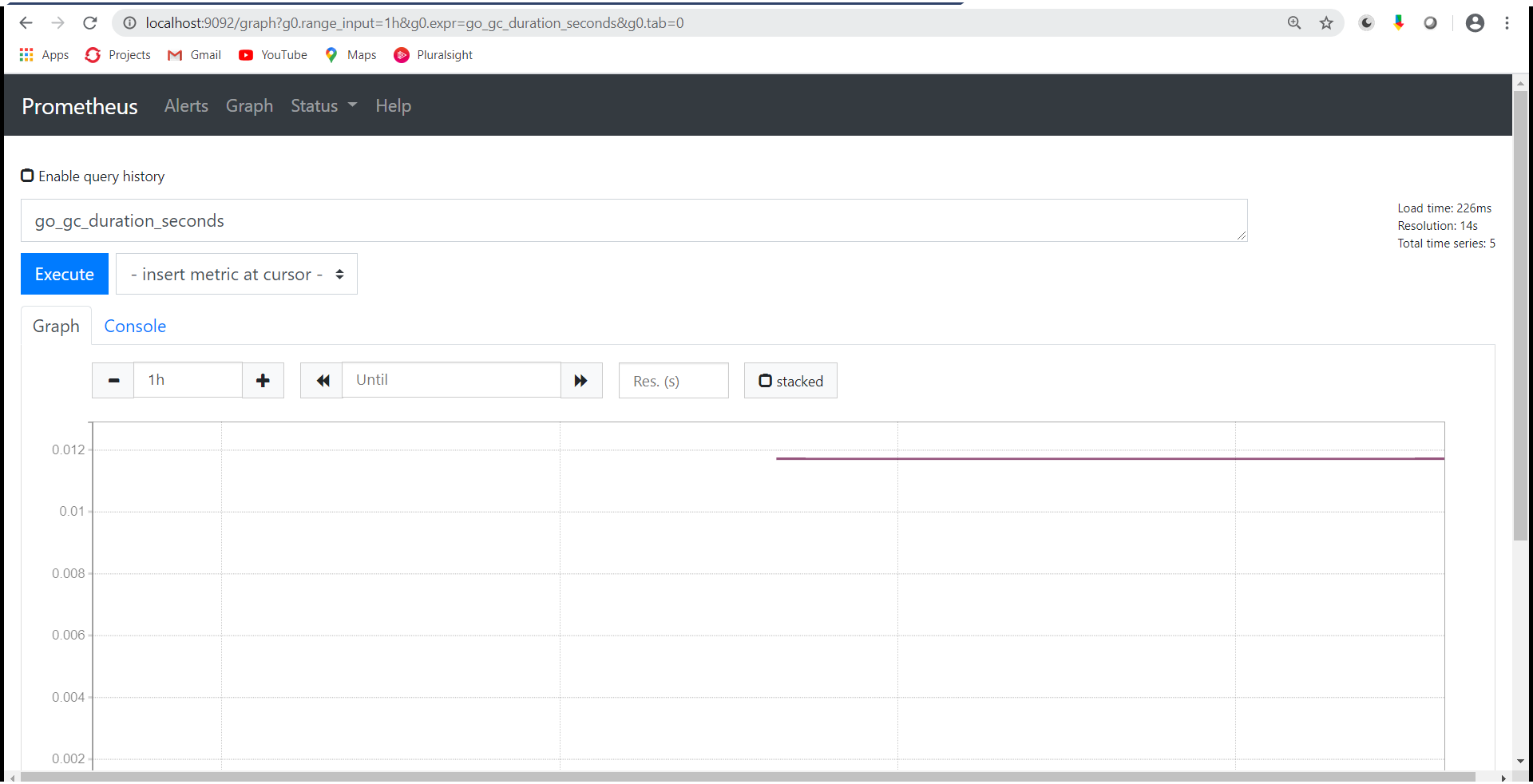
- prometheus.exe --config.file="prometheus.yml" --web.listen-address=:9092
- Refer project
- D:\git\ericmay2021repo\productdockerapi

Prometheus Web UI & Visualizations

- The Prometheus web app includes a built-in utility.
- It describes as an “expression browser” which provides direct access to enter any expression and visualize its result either in a table or graphed over time.

Prometheus


Microservices


localhost:9092/graph?g0.range_input=1h&g0.expr=go_gc_duration_seconds&g0.tab=0
Apps
Projects
Gmail
YouTube
Maps
Pluralsight
Prometheus
Alerts
Graph
Status
Help
Enable query history
go_gc_duration_seconds
Load time: 226ms
Resolution: 14s
Total time series: 5
Execute
- insert metric at cursor -
Graph
Console
1h
Until
Res. (s)
stacked
0.012
0.01
0.008
0.006
0.004
0.002

Custom Metrics

- We can also define some custom metrics.
- The demo contains a Scheduler class which periodically runs the included scheduling Task method.
- To be able to send custom metrics we need to import Meter Registry from the Micrometer library and inject it into our class.

Custom Metrics

- It is possible to instantiate these types of meters from MeterRegistry:
  - Counter: reports merely a count over a specified property of an application
  - Gauge: shows the current value of a meter
  - Timers: measures latencies or frequency of events
  - DistributionSummary: provides distribution of events and a simple summary
  - **Refer implementation of a counter and a gauge for demonstration purposes**

Prometheus Query

- Counters
  - Counter metrics are used for measurements that only increase.
  - Therefore, they are always cumulative—their value can only go up.
  - The only exception is when the counter is restarted, in which case its value is reset to zero.
  - The actual value of a counter is not typically very useful on its own.
  - A counter value is often used to compute the delta between two timestamps or the rate of change over time.

Prometheus Query

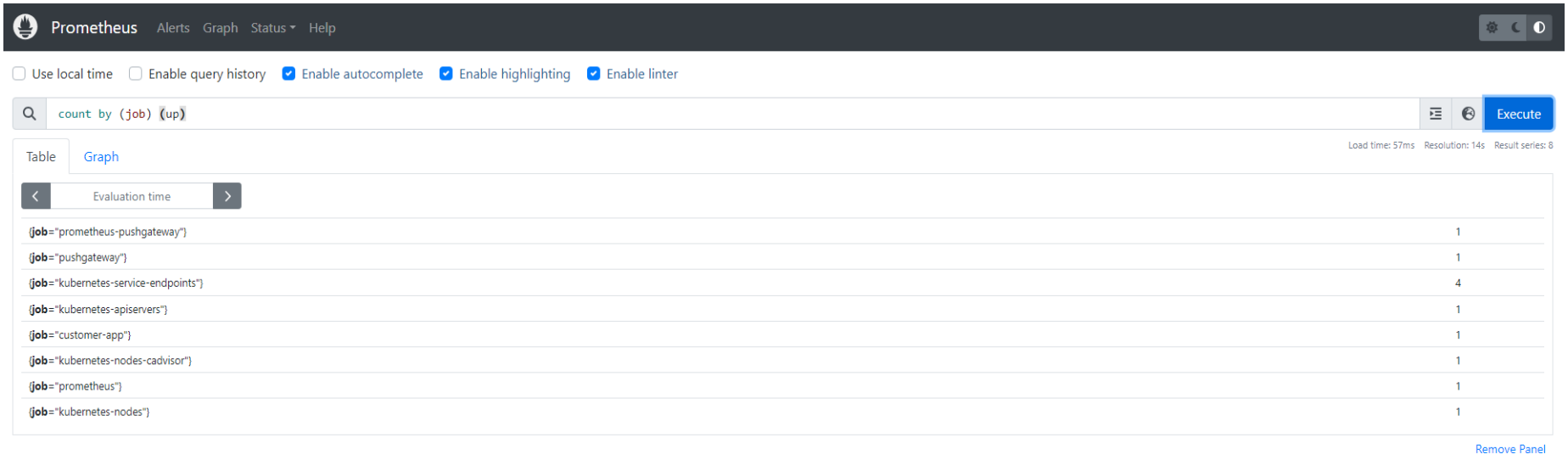
- Counters
- The PromQL query below calculates the average requests per second over the last five minutes
  - rate(prometheus_http_requests_total[5m])
- To calculate the absolute change over a time period, we would use a delta function which in PromQL is called increase():
  - increase(prometheus_http_requests_total[5m])

Prometheus Query

- Counters
- This would return the total number of requests made in the last five minutes, and it would be the same as multiplying the per second rate by the number of seconds in the interval (five minutes in our case):
- rate(prometheus_http_requests_total[5m])*5*60
- count_over_time(process_cpu_usage{instance="customer-app:7074"}[10m])

Prometheus Query

Count by (job) (up)



Prometheus Query

- Gauges
  - Gauge metrics are used for measurements that can arbitrarily increase or decrease.
  - the actual value with no additional processing is meaningful and they are often used.
  - For example, metrics to measure temperature, CPU, and memory usage, or the size of a queue are gauges.

Prometheus Query

- avg_over_time(application_ready_time_seconds{instance="customer-app:7074"}[5m])

Prometheus Query

- **Histograms**
  - Histogram metrics are useful to represent a distribution of measurements.
  - They are often used to measure request duration or response size.
  - Histograms divide the entire range of measurements into a set of intervals—named buckets—and count how many measurements fall into each bucket.

Prometheus Query

- **Histograms**
  - A histogram metric includes a few items:
  - A counter with the total number of measurements.
  - The metric name uses the _count suffix.
  - A counter with the sum of the values of all measurements.
  - The metric name uses the _sum suffix.
  - The histogram buckets are exposed as counters using the metric name with a _bucket suffix and a le label indicating the bucket upper inclusive bound.
  - Buckets in Prometheus are inclusive, that is a bucket with an upper bound of N (i.e., le label) includes all data points with a value less than or equal to N.

Prometheus Query

- **Histograms**
  - http_server_requests_seconds_count{instance="customer-app:7074"}
  - prometheus_http_request_duration_seconds_bucket{instance="localhost:9090"}

Grafana

Open source Solutions Learn Company Downloads Contact us Sign in

If you are using grafana-cli, execute the following command to install the plugin

Shell

```
grafana-cli plugins install yesoreyeram-boomtheme-panel
```

or for specific versions

Shell

```
grafana-cli --pluginUrl https://github.com/yesoreyeram/yesoreyeram-boomtheme-panel/releases/download/v0.2.1/yesor
```

Using helm chart

If you use helm chart to provision grafana, use the following config to install the plugin

YAML

```
plugins:
  - yesoreyeram-boomtheme-panel
```

or for any specific versions

Grafana

Administrator: Command Prompt - docker exec -it prometheus_grafana_1 /bin/bash

```
1784f8db487a:/usr/share/grafana/public/sass$ cd ..
1784f8db487a:/usr/share/grafana/public$ cd ..
1784f8db487a:/usr/share/grafana$ ls
bin     conf    public
1784f8db487a:/usr/share/grafana$ cd bin
1784f8db487a:/usr/share/grafana/bin$ ls
grafana            grafana-cli        grafana-cli.md5     grafana-server      grafana-server.md5  grafana.md5
1784f8db487a:/usr/share/grafana/bin$ grafana-cli plugins install yesoreyeram-boomtheme-panel
Deprecation warning: The standalone 'grafana-cli' program is deprecated and will be removed in the future. Please update all uses of 'g
rafana-cli' to 'grafana cli'
✔Downloaded and extracted yesoreyeram-boomtheme-panel v0.2.1 zip successfully to /var/lib/grafana/plugins/yesoreyeram-boomtheme-panel

Please restart Grafana after installing or removing plugins. Refer to Grafana documentation for instructions if necessary.

1784f8db487a:/usr/share/grafana/bin$
```

Grafana

https://volkovlabs.io/blog/how-to-customize-the-grafana-user-interface-8d70a42dc2b6/

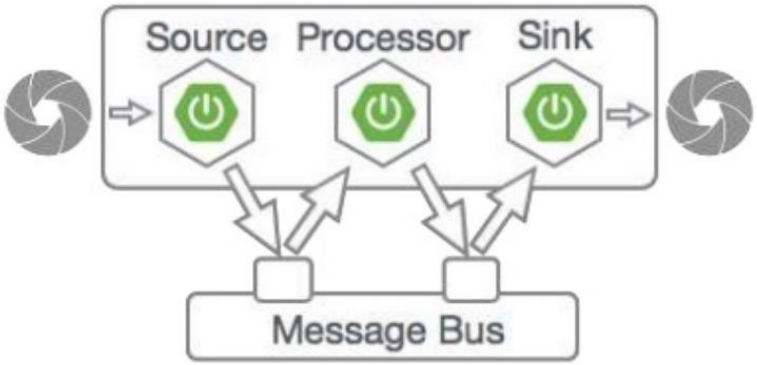
Spring Data Flow

Spring Data Flow

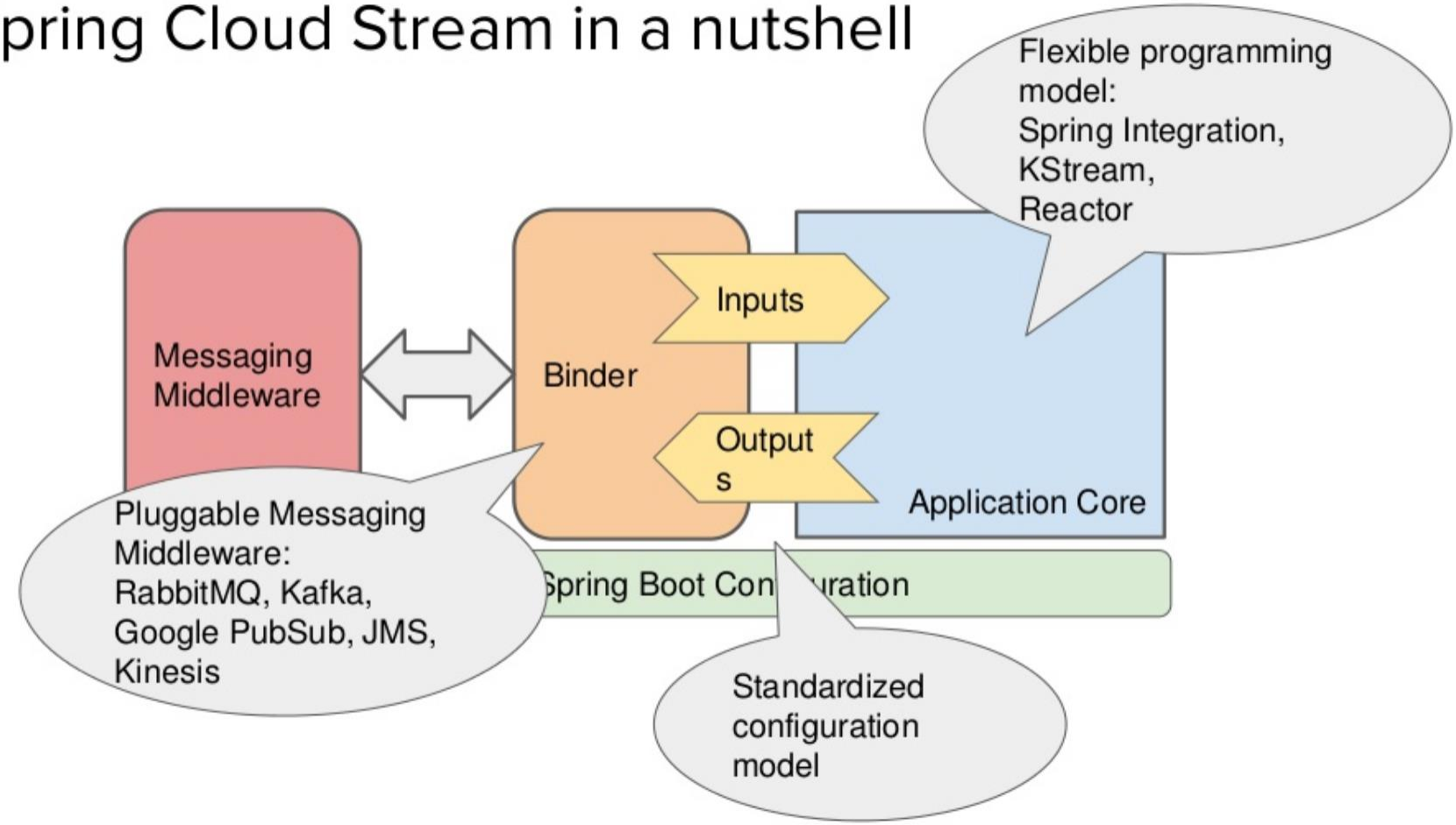
- Spring Cloud Data Flow is a toolkit to build real-time data integration and data processing pipelines by establishing message flows between Spring Boot applications that could be deployed on top of different runtimes.
- Long lived applications require Stream Applications while Short lived applications require Task Applications.
- In this example we make use of Stream Applications. Previously we had already developed Spring Cloud Stream applications to understand the concept of Spring Cloud Stream Source and Spring Cloud Sink and their benefit.

Microservices are not just for web apps

- Spring Cloud Stream
  - Event driven based Microservices
  - Loose coupling via pub/sub messaging
  - Opinionated Primitives
  - For **integration** and **stream processing** use-cases



Spring Cloud Stream in a nutshell



Microservices are not just for web apps

- Spring Cloud Task
  - Tasks are short lived Boot Microservices
  - System tracks invocations, exit-status
  - Spring Batch Jobs are wrapped as Tasks
  - Useful for ETL between databases and filesystems

Spring Data Flow

- 1) Applications:
- Applications will be of two types.
- A. Short lived applications:
- this type of applications can be run for short amount of time and then it will terminate. Example: Spring Cloud Task, it will for whenever we try to execute the task it will launched and then it terminate after the task, same for batch applications.

Spring Data Flow

- 1) Applications:
- B. Long lived applications:
- this type of applications will be run for unbounded amount of time, majorly these applications are message based applications.
- Example:
- Spring Cloud Stream, this type of applications will consume messages and produce messages to data flow server configured message broker.

Spring Data Flow

- 2) Application packaging:
- the target run time will be of two types.
- A. Spring boot jar file:
- we can specify either spring boot jar or any http URL pointing to spring boot jar file or maven artifact.

- B. Docker Image:
- we can also specify the docker image url for micro service applications.

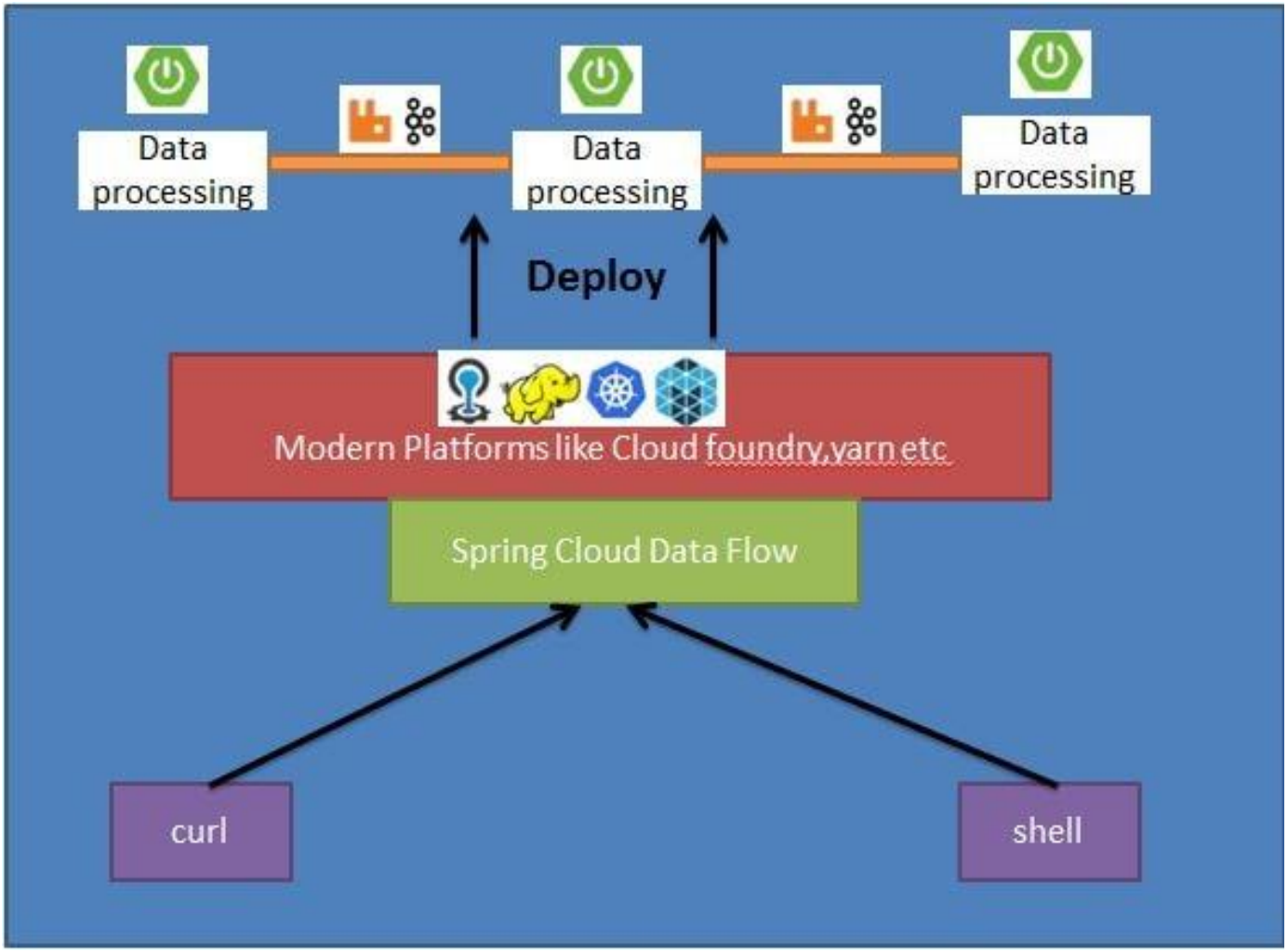
Spring Data Flow

- 3) Runtime:
- the target system run time where our applications will be executed. The list of supported run times are:
- Cloud Foundry Apache YARN Kubernetes Apache Mesos Local Server for development.

Spring Data Flow

- Microservice based Streaming and Batch data processing for Cloud Foundry and Kubernetes.
- Spring Cloud Data Flow provides tools to create complex topologies for streaming and batch data pipelines.
- The data pipelines consist of Spring Boot apps, built using the Spring Cloud Stream or Spring Cloud Task microservice frameworks.
- Spring Cloud Data Flow supports a range of data processing use cases, from ETL to import/export, event streaming, and predictive analytics.

Spring Data Flow



Source	Processor	Sink
file	aggregator	aggregate-counter
ftp	bridge	cassandra
gemfire	filter	counter
gemfire-cq	groovy-filter	field-value-counter
http	groovy-transform	file
jdbc	header-enricher	ftp
jms	httpclient	gemfire
load-generator	pmml	gpfdist
loggregator	python-http	hdfs
mail	python-jython	hdfs-dataset
mongodb	scriptable-transform	jdbc
Mqtt	splitter	log

Rabbit	tasklaunchrequest-transform	mongodb
S3	tcp-client	mqtt
sftp	tensorflow	pgcopy
syslog	transform	rabbit
tcp	twitter-sentiment	redis-pubsub
tcp-client		router
time		s3
trigger		sftp
triggertask		task-launcher-cloudfoundry
twitterstream		task-launcher-local
		task-launcher-yarn
		tcp
		throughput
		websocket

Spring Data Flow

- The Spring Cloud Data Flow server uses Spring Cloud Deployer, to deploy data pipelines made of Spring Cloud Stream or Spring Cloud Task applications onto modern platforms such as Cloud Foundry and Kubernetes.

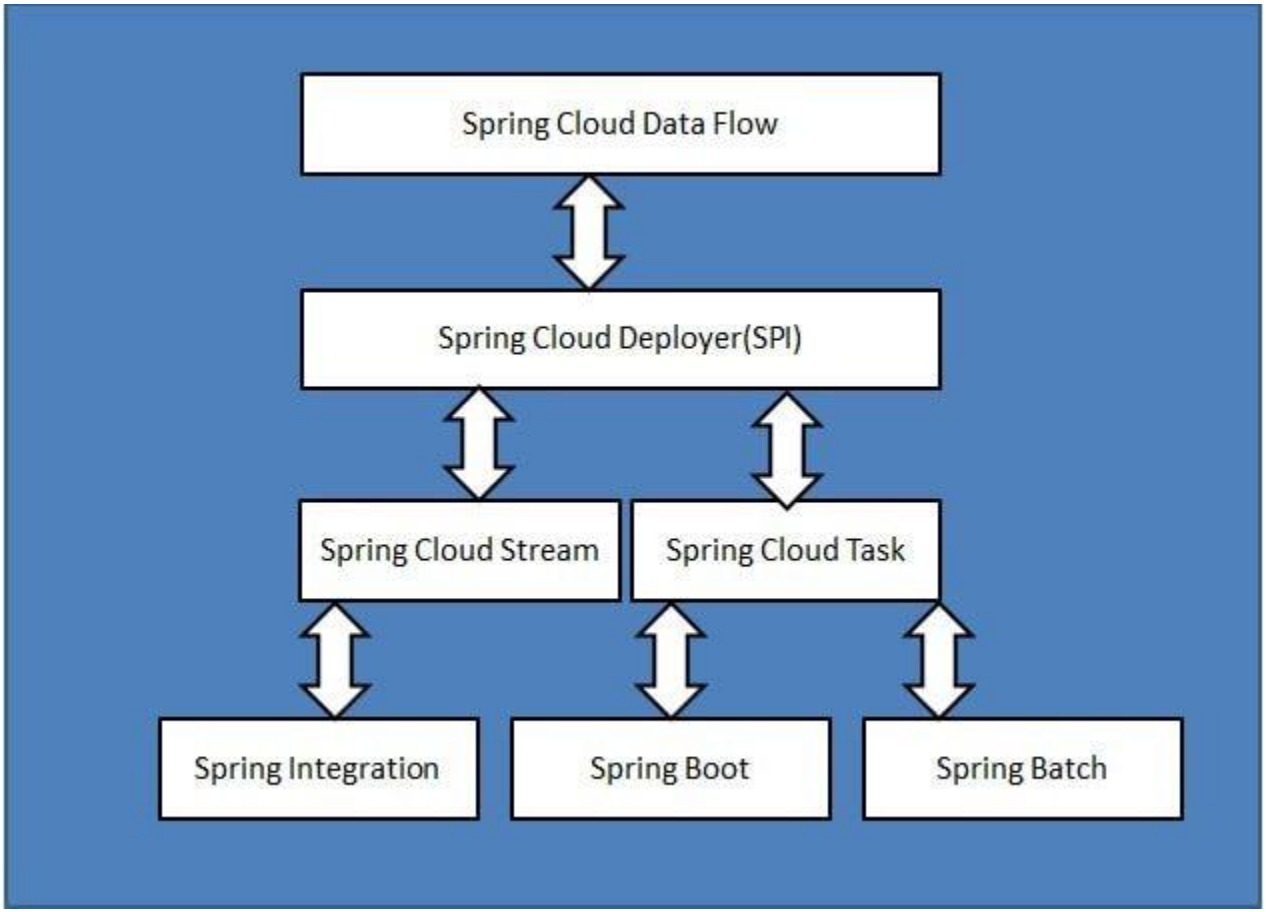
- Custom stream and task applications, targeting different middleware or data services, can be built using the familiar Spring Boot style programming model.

Spring Data Flow

- Pipelines consist of Spring Boot apps, built using the Spring Cloud Stream or Spring Cloud Task microservice frameworks.
- SCDF can be accessed using the REST API exposed by it or the web UI console.
- We can make use of metrics, health checks, and the remote management of each microservice application .
- Also we can scale stream and batch pipelines without interrupting data flows.
- With SCDF we build data pipelines for use cases like data ingestion, real-time analytics, and data import and export.

Spring Data Flow

- SCDF is composed of the following Spring Projects-



Spring Data Flow

- java -jar spring-cloud-dataflow-server-local-1.3.0.M1.jar

- java -jar spring-cloud-dataflow-shell-1.3.0.M1.jar

Spring data flow

- app register --name myprocessor --type processor --uri file:///F:/virtusa_microservices_oct2019/source-0.0.1-SNAPSHOT.jar

dataflow 1.3.0.M1

```
               0 Dir(s)  66,305,060,864 bytes free

F:\virtusa_microservices_oct2019>java -jar spring-cloud-dataflow-shell-1.3.0.M1.jar

  ____                              ____ _                __
 / ___| _ __  _ __(_)_ __   __ _   / ___| | ___  _   _  __| |
 \___ \| '_ \| '__| | '_ \ / _` | | |   | |/ _ \| | | |/ _` |
  ___) | |_) | |  | | | | | (_| | | |___| | (_) | |_| | (_| |
 |____/| .__/|_|  |_|_| |_|\__, |  \____|_|\___/ \__,_|\__,_|
  ____ |_|    _          __|___/                 __________
 |  _ \  __ _| |_ __ _  |  ___| | _____      __  \ \ \ \ \ \
 | | | |/ _` | __/ _` | | |_  | |/ _ \ \ /\ / /   \ \ \ \ \ \
 | |_| | (_| | || (_| | |  _| | | (_) \ V  V /    / / / / / /
 |____/ \__,_|\__\__,_| |_|   |_|\___/ \_/\_/    /_/_/_/_/_/

1.3.0.M1

Welcome to the Spring Cloud Data Flow shell. For assistance hit TAB or type "help".
dataflow:>
```

dataflow 1.3.0.M1

```
dataflow:>app register --name source-app --type source --uri file:///F:/virtusa_microservices_oct2019/source-0.0.1-SNAPSHOT.jar
Successfully registered application 'source:source-app'
dataflow:>app register --name processor-app --type processor --uri file:///F:/virtusa_microservices_oct2019/processor-0.0.1-SNAPSHOT.jar
Successfully registered application 'processor:processor-app'
dataflow:>app register --name sink-app --type sink --uri file:///F:/virtusa_microservices_oct2019/sink-0.0.1-SNAPSHOT.jar
Successfully registered application 'sink:sink-app'
dataflow:>stream create --name log-data --definition 'source-app|processor-app|sink-app'

Command failed org.springframework.cloud.dataflow.rest.client.DataFlowClientException: Cannot create stream log-data because another one has already been created with the same name

dataflow:>stream create --name log-data --definition 'source-app|processor-app|sink-app'

Created new stream 'log-data'
dataflow:>stream deploy --name log-data
Deployment request has been sent for stream 'log-data'
dataflow:>
```

dataflow 1.3.0.M1

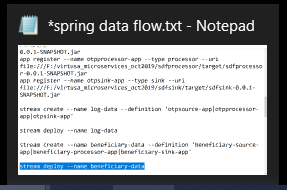
```
| beneficiary-source-app |  |  |  |

dataflow:>app register --name beneficiary-processor-app --type processor --uri file:///F:/virtusa_microservices_apr2020/beneficiaryproc
essor/target/beneficiaryprocessor-0.0.1-SNAPSHOT.jar
Successfully registered application 'processor:beneficiary-processor-app'
dataflow:>app list
```

source	processor	sink	task
beneficiary-source-app	beneficiary-processor-app		

```
dataflow:>app register --name beneficiary-sink-app --type sink --uri file:///F:/virtusa_microservices_apr2020/beneficiarysink/target/be
neficiarysink-0.0.1-SNAPSHOT.jar
Successfully registered application 'sink:beneficiary-sink-app'
dataflow:>app list
```

source	processor	sink	task
beneficiary-source-app	beneficiary-processor-app	beneficiary-sink-app	

```
dataflow:>stream create --name beneficiary-data --definition 'beneficiary-source-app|beneficiary-processor-app|beneficiary-sink-app'
Created new stream 'beneficiary-data'
dataflow:>
dataflow:>stream deploy --name beneficiary-data
Deployment request has been sent for stream 'beneficiary-data'
dataflow:>
```



Type here to search

ENG 21:44 21/05/2020

Spring Cloud Tutorial - Stream P | 6 Practical Uses for a Microservic | Download the jar using http://re | Spring Cloud Data Flow

localhost:9393/dashboard/#/apps

Apps | Insert title here | Empire | New Tab | How to use Asserti... | Browser Automatio... | node.js - How can I... | Freelancer-dev-810... | Courses | New Tab | hi | Airtel 4G Hotspot | nt8F83

spring

Apps | Runtime | Streams | Tasks | Jobs | Analytics | About

Apps

This section lists all the available applications and provides the control to register/unregister them (if applicable).

+ REGISTER APPLICATION(S) | NO APP SELECTED TO UNREGISTER | BULK IMPORT APPLICATIONS | Filter items

Name	Type	URI	Actions
source-app	source	maven://com.virtusa.source:source-0.0.1-SNAPSHOT.jar	
processor-app1	processor	maven://com.virtusa.processor:processor-0.0.1-SNAPSHOT.jar	
sink1-app	sink	maven://com.virtusa.sink:sink-0.0.1-SNAPSHOT.jar	

« Previous 1 Next »

PROJECT
Project Page
Issue Tracker

DOCUMENTATION
Docs
Sources
Api Docs

NEED HELP?
For questions + support:
Stackoverflow

Type here to search

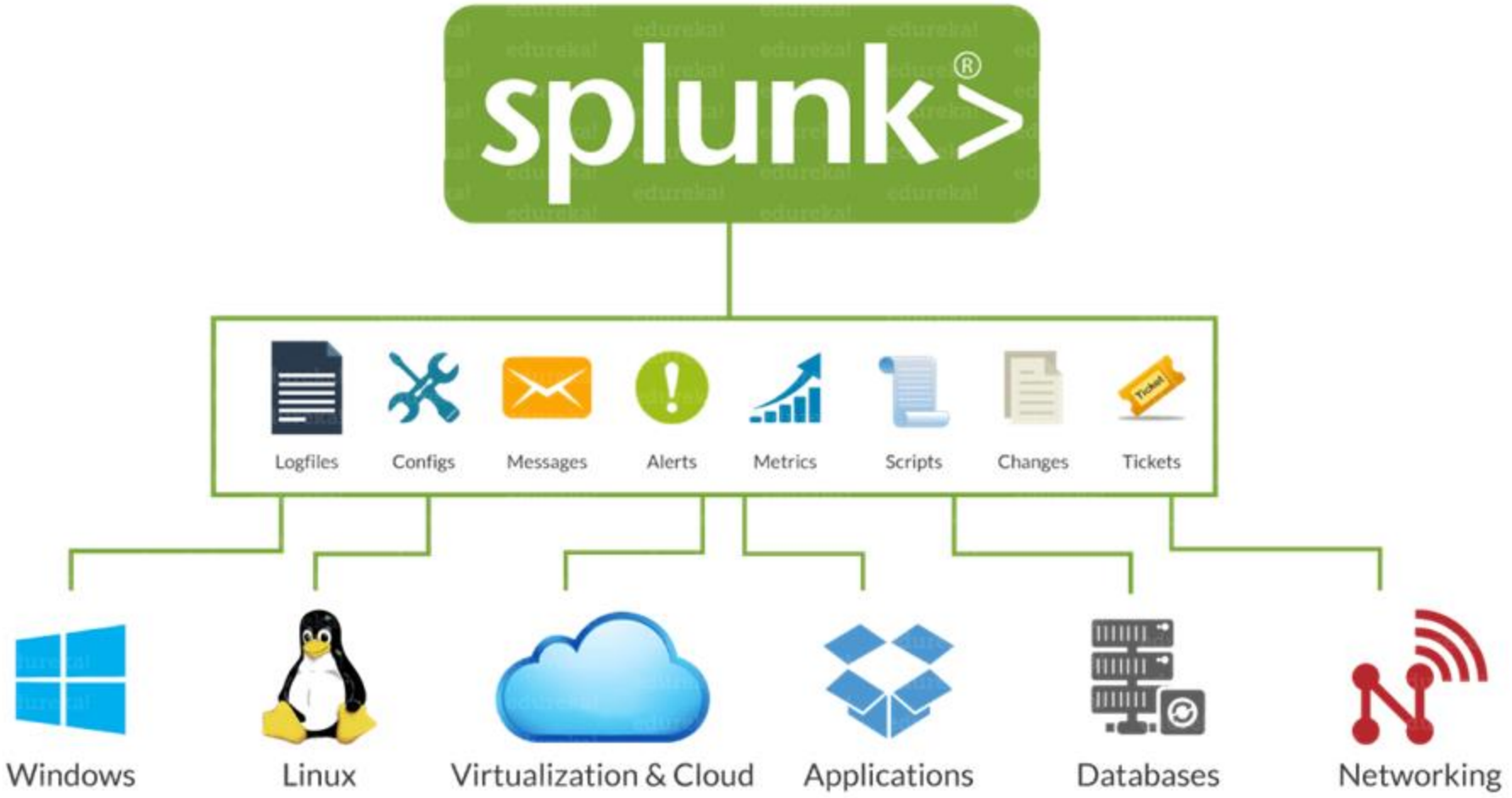
ENG 06:37 05/11/2019

- stream create --name log-data --definition 'source-app | processor-app| sink-app‘
- stream deploy --name log-data
- Go to Spring Cloud Data Flow UI Console - http://localhost:9393/dashboard

Splunk

- Splunk is a one-stop solution as it automatically pulls data from various sources and accepts data in any format such as .csv, json, config files, etc.
- Splunk is the easiest tool to install and allows functionality like: searching, analyzing, reporting as well as visualizing machine data.
- It has a huge market in the IT infrastructure and business.
- Many big players in the industry are using Splunk such as Dominos, Adobe, Bosch, Vodafone, Coca-Cola etc.

Splunk

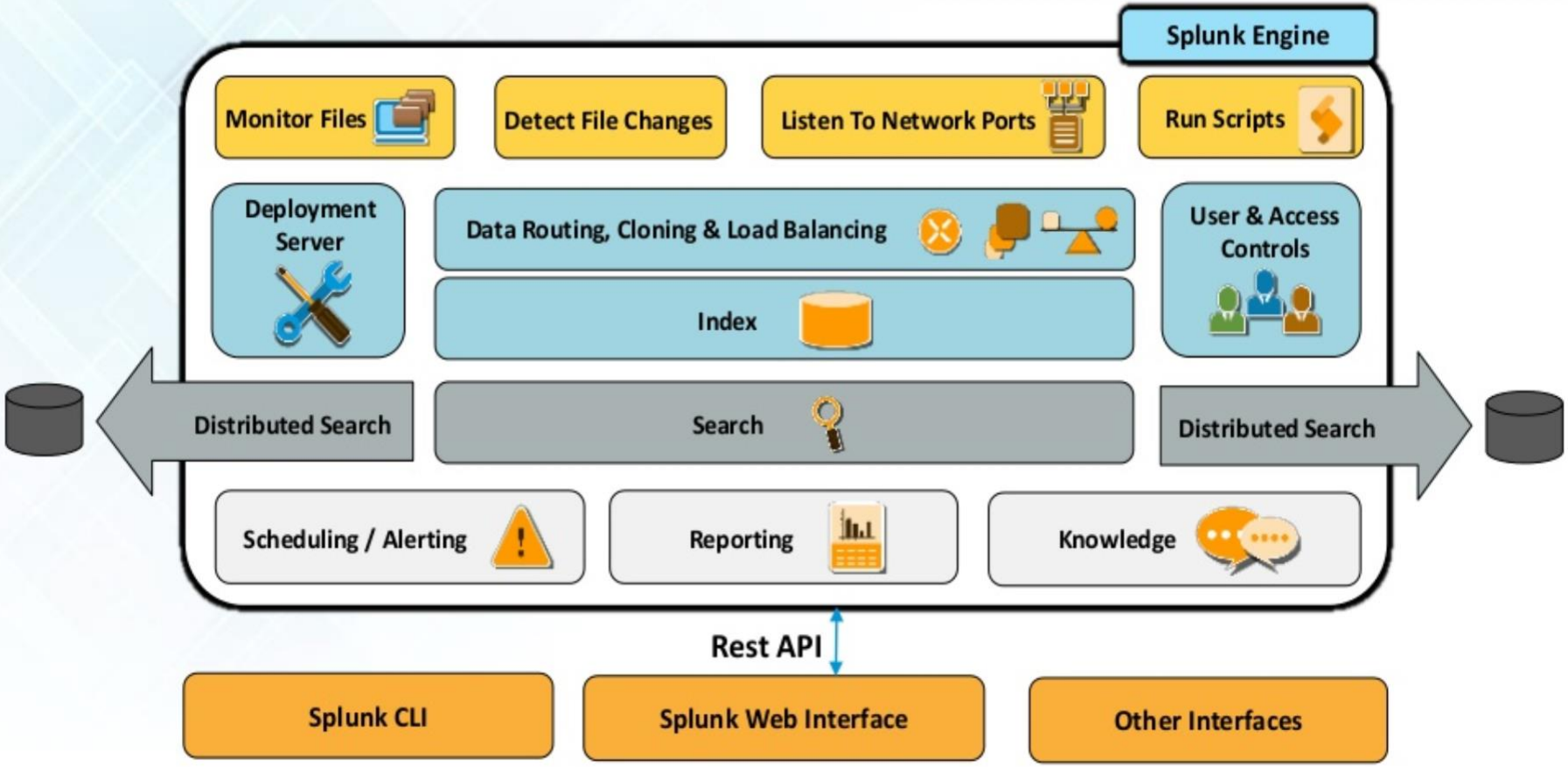

splunk>
Logfiles
Configs
Messages
Alerts
Metrics
Scripts
Changes
Tickets
Windows
Linux
Virtualization & Cloud
Applications
Databases
Networking

Splunk

- Splunk collects data in real-time from multiple systems.
- It accepts data in any form, example- log file, .csv, json, config etc.
- Splunk can pull data from database, cloud and any other OS
- It analyze and visualize the data for better performance.
- Splunk give alerts/ event notifications.
- Provides real-time visibility.
- It satisfies industry needs like horizontal scalability (using many systems in parallel)

Splunk Architecture

Microservices



Splunk Architecture

- Splunk CLI/ splunk web interface or any other interface interacts with the search head.
- This communication happens via Rest API.
- It can be used to search head to make distributed searches, setup knowledge objects for operational intelligence, perform scheduling/ alerting and create reports or dashboards for visualization.
- It can also be used to run scripts for automating data forwarding from remote Splunk forwarders to pre-defined network ports.
- After that you can monitor the files that are coming at real time and analyze if there are any anomalies and set alert/ reminders accordingly.
- It can be used to perform routing, cloning and load balancing of the data that is coming in from the forwarder, before they are stored in an indexer.
- It can also be used to create multiple users to perform various operations on the indexed data.

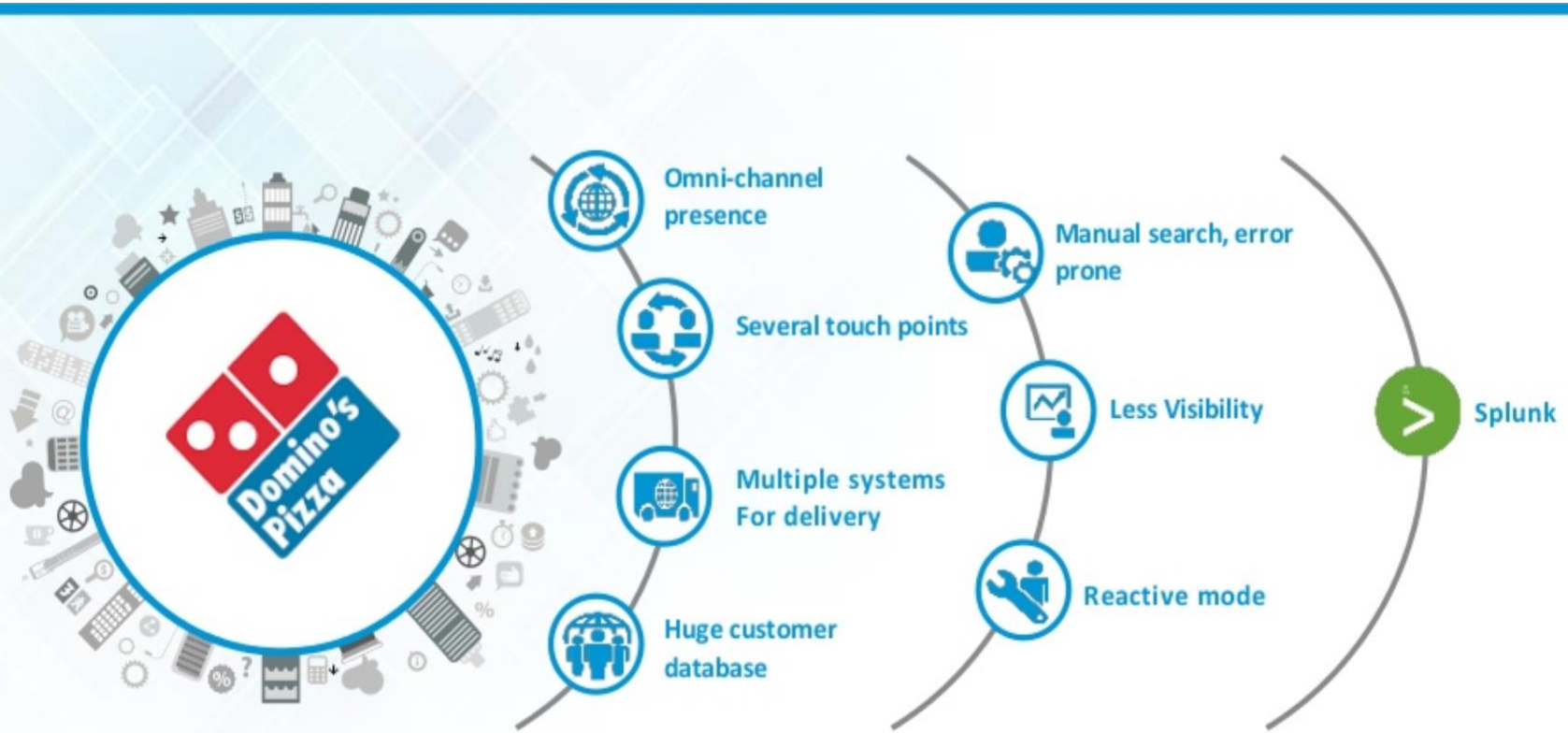
Splunk Users

Persona	Industry Role	Activities
Administrator	network engineer, system administrator	• Configures, administers, optimizes, and secures the Splunk Enterprise deployment. • Sets up user accounts and permissions. • Gets data into Splunk Enterprise.
Knowledge Manager	data analyst, system administrator	• Oversees knowledge object creation, normalization, and usage across teams, departments, and deployments. • Gets the data into Splunk Enterprise, or works with the administrator to do so. • Creates and shares data models.
Search User	data analyst, IT professional, network engineer, security analyst, system administrator	• Uses Search to investigate server problems, understand configurations, monitor user activities, and troubleshoot escalated problems. • Builds reports and dashboards to monitor the health, performance, activity, and capacity of their IT infrastructure. • Identifies patterns and trends that are indicators of routine problems.
Pivot User	business professional, data analyst, executive, IT professional, manager, system administrator	• Uses Pivot to build reports based on data models created by the Knowledge Manager. • Creates reports and dashboards to monitor their businesses. • Identifies trends in the health and performance of their businesses.
Developer	system integrator, professional developer	• Integrates data and functionality of applications with Splunk Enterprise. • Builds Splunk apps and add-ons with custom dashboards and data visualizations.

Splunk vs Other Tools

Features	Splunk	Sumo Logic	ELK
Searching	✓	✓	Only possible with Integrations
Analysis	✓	✓	Only possible with Integrations
Visualization Dashboard	✓	✓	Only possible with Integrations
SaaS Setup	✓	✓	✓
On Premise Setup	✓	✗	✓
Input any data type	✓	✓	Needs plugins
Plugins & Integration	✓	✓	✓
Customer Support	✓	Available; but not proficient	Available; but not proficient
Documentation & Community	✓	✗	✓

Case: Domino's



Case: Domino's

Interactive map

- Shows all the orders coming from across US in real time
- Brought employee satisfaction

Real-time Feedback

- Employees constantly see what customers are saying
- Helped them understand customer expectations

Dashboard

- Used to keep score and set targets
- Compare performance with previous week

Payment Process

- Analysed the speed of different payment modes
- Determine error free payments modes

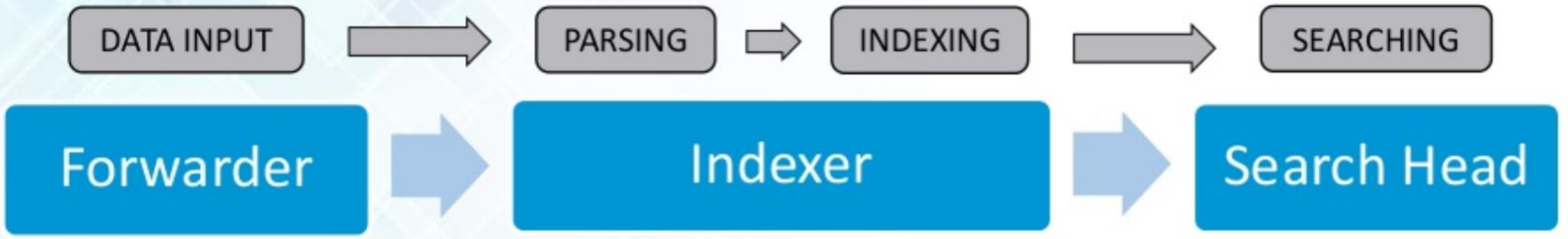
Promotional Support

- Track how various promotional offers are impacting in real-time
- Initially, determining the impact of promotions took almost a day

Performance Monitor

- Monitor the performance of Domino's in-house developed point of sales systems

Splunk Components

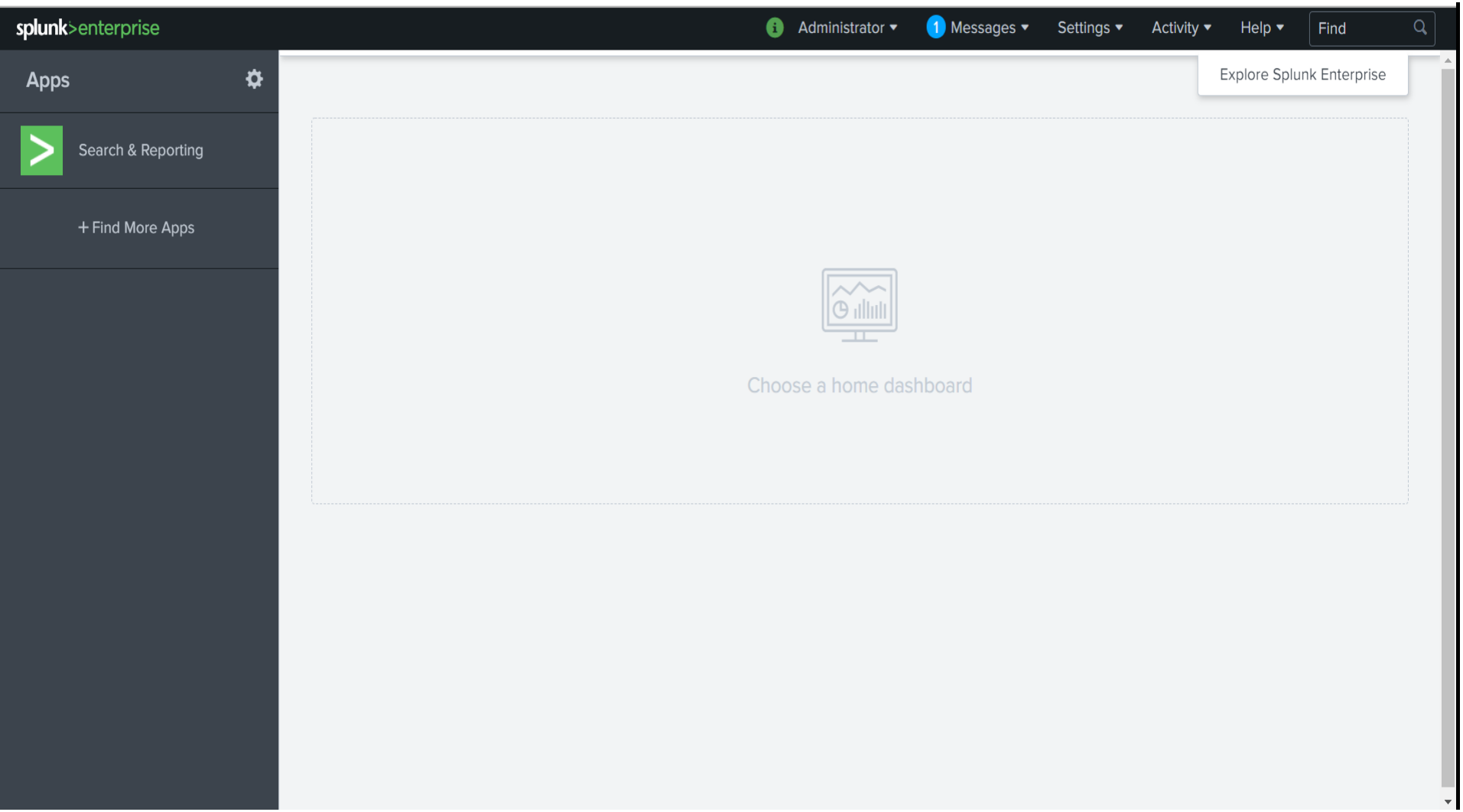


- Collects the data from remote machines
- Forwards the data to the Indexer in real-time

- Processes the incoming data in real-time
- Stores & Indexes the data on disk

- End users interact with Splunk through Search Head
- Allows users to do searching, analysis & visualization


Microservices


splunk>enterprise
Administrator
1 Messages
Settings
Activity
Help
Find
Explore Splunk Enterprise
Apps
Search & Reporting
+ Find More Apps
Choose a home dashboard

Search | Splunk 8.0.8 | localhost:7070/index?name=test | Splunk spring boot log - YouTube

localhost:8200/en-GB/app/search/search

Apps | Insert title here | Empire | New Tab | How to use Asserti... | Browser Automatio... | node.js - How can I... | Freelancer-dev-810... | Courses | New Tab | Airtel 4G Hotspot | nt8F83 | Reading list

splunk>enterprise App: Search & Reporting | Administrator | 1 Messages | Settings | Activity | Help | Find

Search | Analytics | Datasets | Reports | Alerts | Dashboards | Search & Reporting

Search

enter search here... | Last 24 hours

No Event Sampling | Smart Mode

How to Search

If you are not familiar with the search features, or want to learn more, see one of the following resources.

Documentation | Tutorial

What to Search

Waiting for data...

Data Summary

View sources, source types and hosts.

> Search History

localhost:8200/en-GB/app/search/search#

Type here to search | ENG | 22:36 | 18/03/2021

Search | Splunk 8.0.8 | localhost:7070/index?name=test | Splunk spring boot log - YouTube

localhost:8200/en-GB/app/search/search

Apps | Insert title here | Empire | New Tab | How to use Asserti... | Browser Automatio... | node.js - How can I... | Freelancer-dev-810... | Courses | New Tab | Airtel 4G Hotspot | nt8F83 | Reading list

splunk>enterprise App: Search & Reporting Administrator Messages Settings Activity Help Find

Search Analytics Datasets Rep Search & Reporting

Search

enter search here...

Last 24 hours

No Event Sampling Smart Mode

How to Search

If you are not familiar with the search feat... resources.

Documentation Tutorial

INDEXED EARLIEST EVENT LATEST EVENT

10 minutes ago

Data Summary

Search History

Data Summary

Hosts (1) Sources Sourcetypes (2)

filter

Host		Count	Last Update
splunkforwarder		38	18/03/2021 16:57:56.000

Type here to search ENG 22:37 18/03/2021

Search | Splunk 8.0.8 | localhost:7070/index?name=test | Splunk spring boot log - YouTube

localhost:8200/en-GB/app/search/search?q=search%20host%3Dsplunkforwarder&sid=1616087257.107&display.page.search.mode=smart&dispatch.sample_ra...

Apps | Insert title here | Empire | New Tab | How to use Asserti... | Browser Automatio... | node.js - How can I... | Freelancer-dev-810... | Courses | New Tab | Airtel 4G Hotspot | nt8F83 | Reading list

splunk>enterprise | App: Search & Reporting | Administrator | 1 Messages | Settings | Activity | Help | Find

Search | Analytics | Datasets | Reports | Alerts | Dashboards | Search & Reporting

New Search

Save As | Close

host=splunkforwarder | Last 24 hours

✓ 38 events (17/03/2021 17:00:00.000 to 18/03/2021 17:07:37.000) | No Event Sampling | Job | Smart Mode

Events (38) | Patterns | Statistics | Visualization

Format Timeline | − Zoom Out | + Zoom to Selection | × Deselect | 1 hour per column

List | Format | 20 Per Page | < Prev | 1 | 2 | Next >

< Hide Fields | All Fields

SELECTED FIELDS
a host 1
a source 2
a sourcetype 2

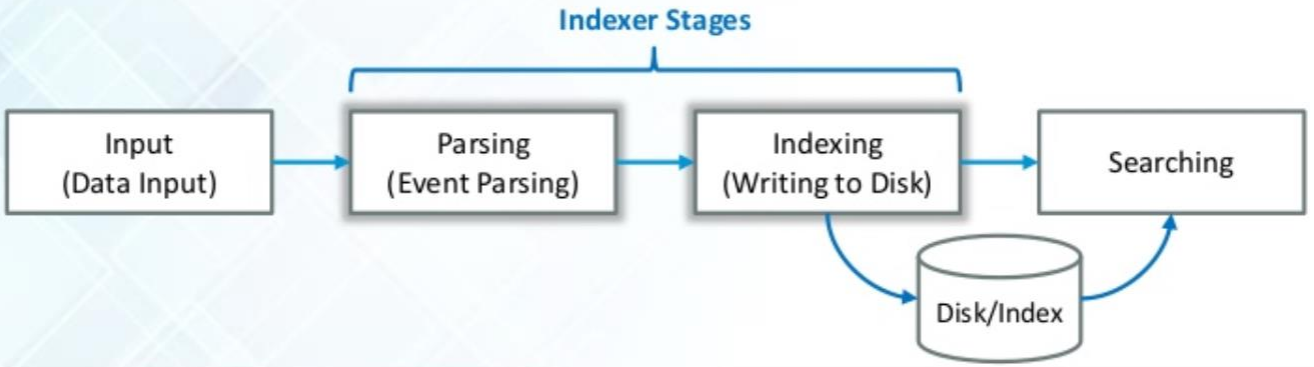
INTERESTING FIELDS
a @timestamp 19
a class 11
date_hour 1
date_mday 1

Time	Event
18/03/2021 16:56:50.554	{ [-] @timestamp: 2021-03-18T16:56:50.554Z class: com.virtusa.DemoController message: name=test parent: service: demo-application severity: INFO span: 97b4b76a5e8394a8 thread: http-nio-7070-exec-10 trace: 97b4b76a5e8394a8 }

Type here to search | ENG | 22:37 18/03/2021

Indexer for Data Storage and Processing

Microservices



In the Parsing stage, only relevant data is converted into events:

- Customer Region
- Order revenue
- Time of purchase
- Device used by customers
- Coupons/ Offers used

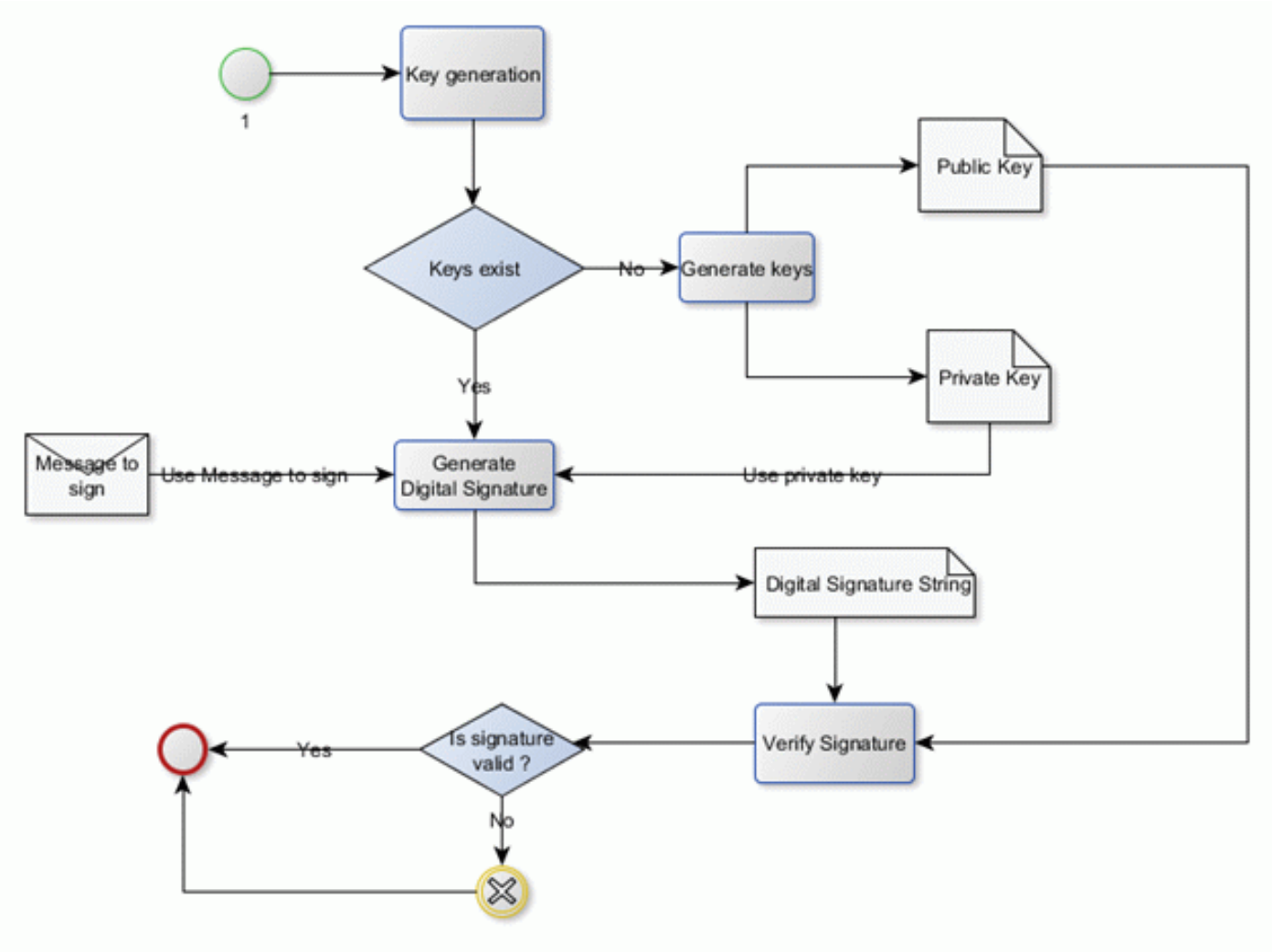
In the Indexing stage, events are sorted and indexed for storage based on:

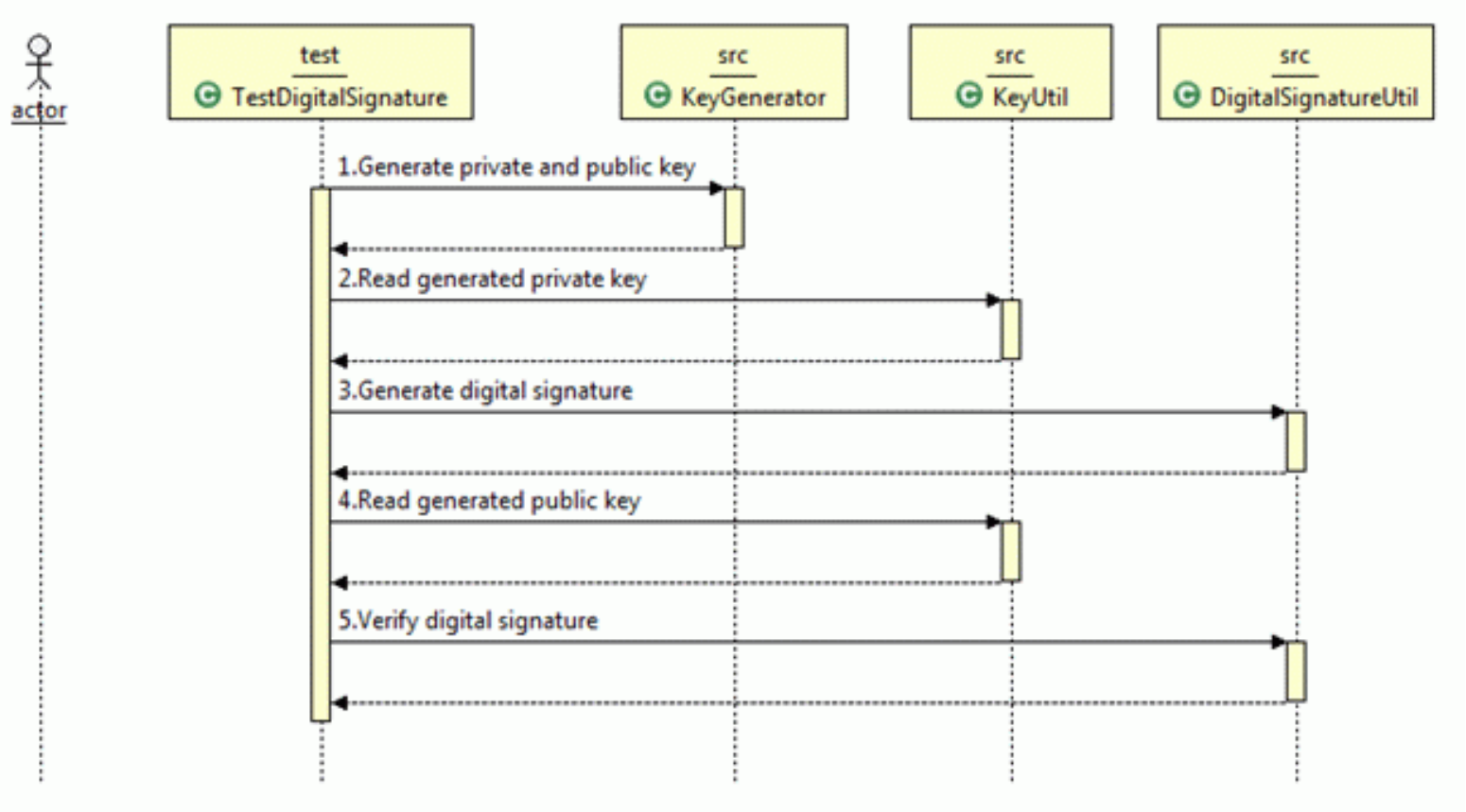
- Sales by Geography
- Order revenue
- Time of purchase
- Device preferred by customers
- Coupons/ Offers used

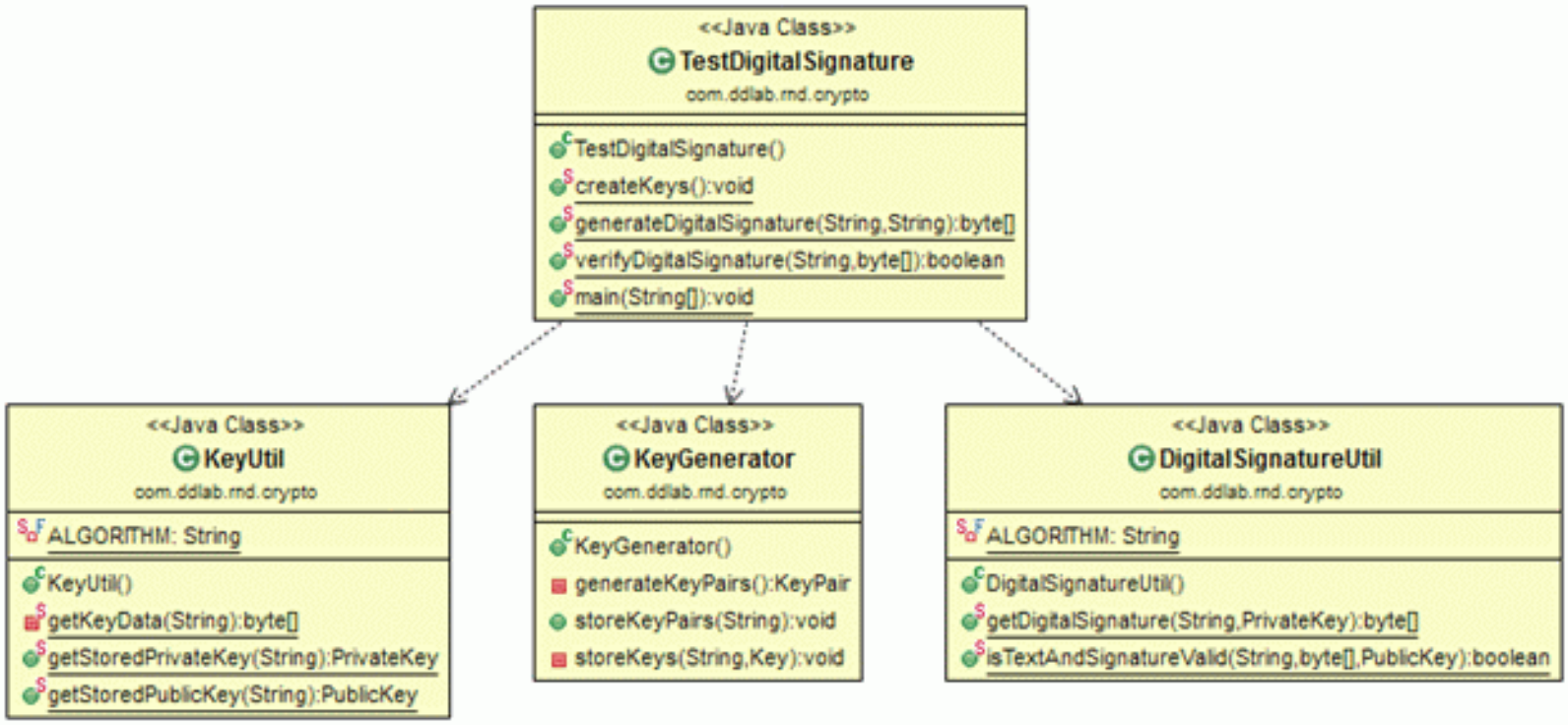
* The details mentioned in this slide are representative in nature and data present might not be accurate.

Digital Signature

- Create a pair of keys called Private and Public keys
- Use the private key and your text message to generate a digital signature
- Send the public key, actual text message and digital signature separately to the destination
- Use the public key, text message and digital signature to verify the message
- If the verification is successful then process the message otherwise throw an exception and discard the message.

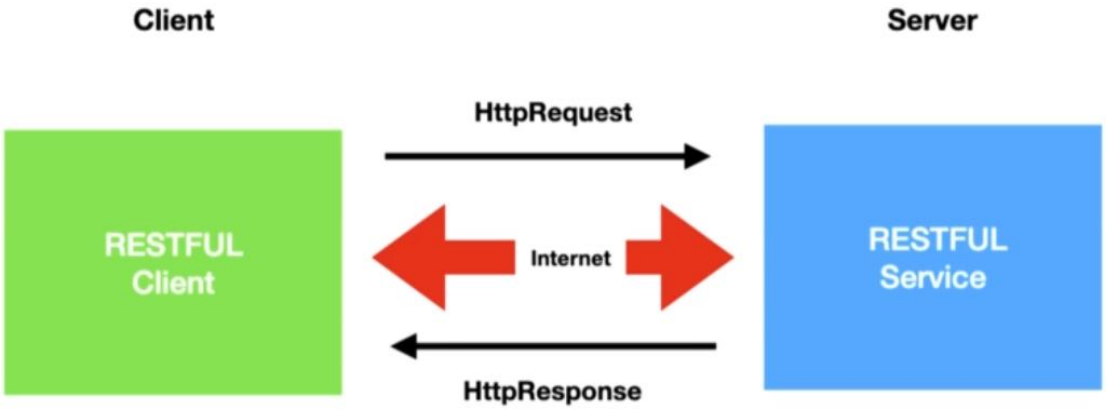

1
Key generation
Keys exist
No
Generate keys
Public Key
Private Key
Yes
Message to sign
Use Message to sign
Generate Digital Signature
Use private key
Digital Signature String
Verify Signature
Is signature valid ?
Yes
No


actor
test
TestDigitalSignature
src
KeyGenerator
src
KeyUtil
src
DigitalSignatureUtil
1.Generate private and public key
2.Read generated private key
3.Generate digital signature
4.Read generated public key
5.Verify digital signature


<<Java Class>>
TestDigitalSignature
com.ddlab.rnd.crypto
TestDigitalSignature()
createKeys():void
generateDigitalSignature(String,String):byte[]
verifyDigitalSignature(String,byte[]):boolean
main(String[]):void
<<Java Class>>
KeyUtil
com.ddlab.rnd.crypto
ALGORITHM: String
KeyUtil()
getKeyData(String):byte[]
getStoredPrivateKey(String):PrivateKey
getStoredPublicKey(String):PublicKey
<<Java Class>>
KeyGenerator
com.ddlab.rnd.crypto
KeyGenerator()
generateKeyPairs():KeyPair
storeKeyPairs(String):void
storeKeys(String,Key):void
<<Java Class>>
DigitalSignatureUtil
com.ddlab.rnd.crypto
ALGORITHM: String
DigitalSignatureUtil()
getDigitalSignature(String,PrivateKey):byte[]
isTextAndSignatureValid(String,byte[],PublicKey):boolean

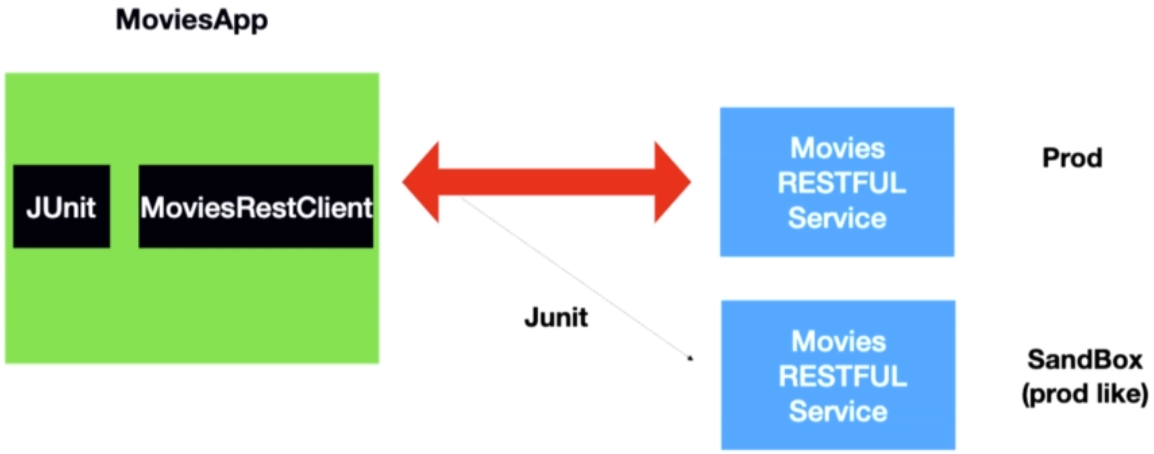
RESTFUL APIs

- Easy to build a functionality and expose it to the world.



Wiremock

Typical RESTFUL Integration



Typical RESTFUL Integration

What can be tested in a **SandBox Environment** ?

- 2XX Response Codes
- 4XX Response Codes

Typical RESTFUL Integration

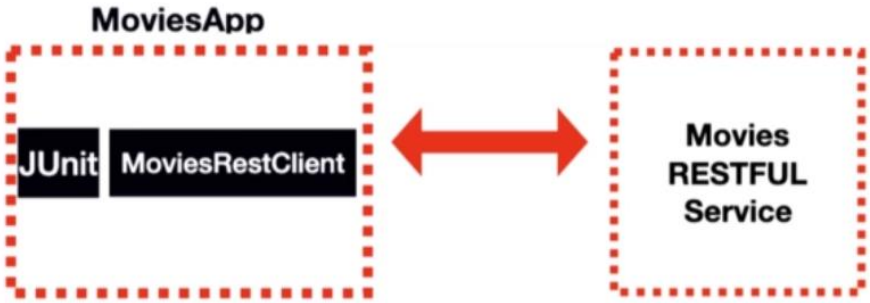
What else needs to be tested for a complete **RESTFUL** Integration?

- 5xx Response Codes (Fault Responses)
- Latency
- Timeouts

WireMock to the rescue!

Challenges in Todays Software Development

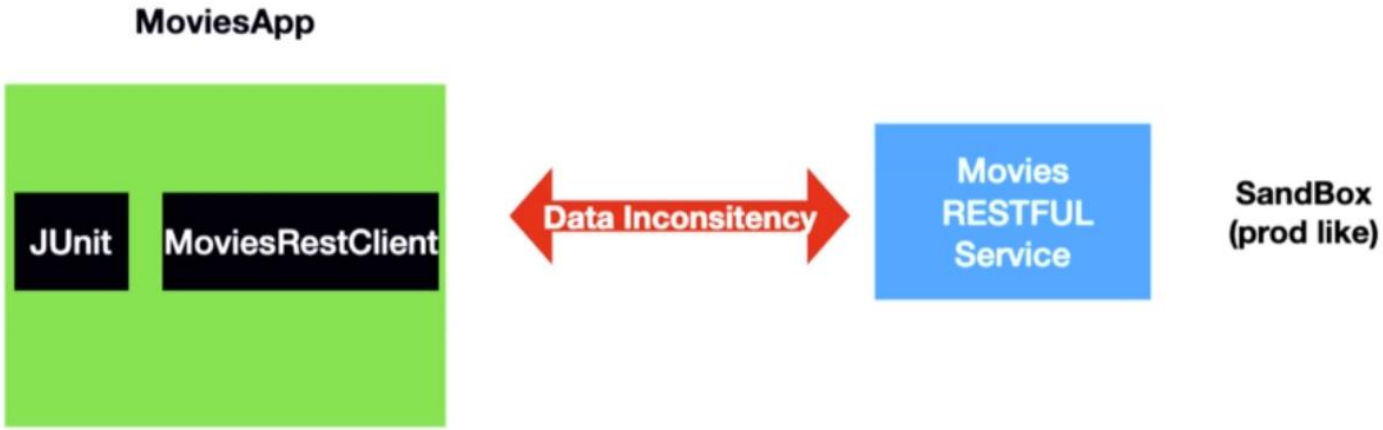
- Development of Client app and Third Party service in parallel.
- Just the contract is finalized.



WireMock to the rescue!

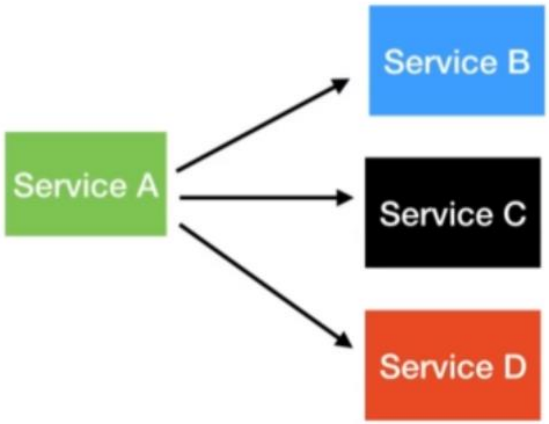
Challenges in JUnit Testing

- Data may be inconsistent in the SandBox environment.



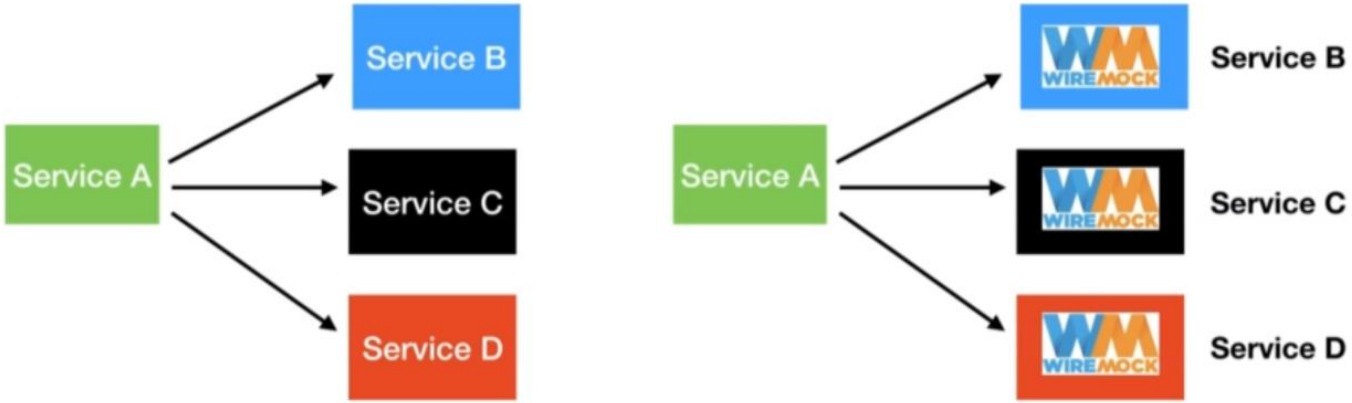
Challenges in JUnit Testing

- **Service Virtualization** in MicroServices environment



Challenges in JUnit Testing

- **Service Virtualization** in MicroServices environment

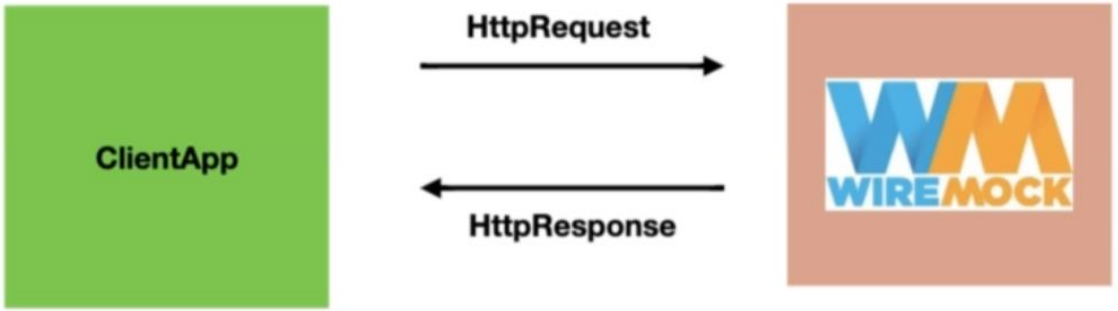


Why Wiremock

- Simulate HTTP responses for a RESTFUL service that does not exist
- Service Virtualization tool
- Static data for the Unit Tests
- Simulate Fault HTTP Responses
- Simulate Timeout
- Simulate Latency

What is WireMock?

- WireMock is a **HTTP API simulator** or **HTTP Mock Web Server**



What is WireMock?

- Configure N number of **Stubs** programmatically
- What is a Stub ?
  - A stub refers to a combination of HttpRequest and HttpResponse.

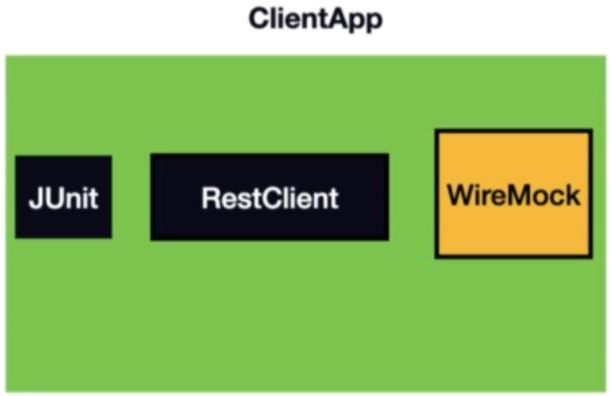
What is WireMock?

- WireMock is built on **Java**
- WireMock should be used only in the scope of Unit/Integration Testing

What is WireMock?

- WireMock can be run in two modes:
  - Embedded
  - Standalone Mode

WireMock - Embedded



- WireMock runs alongside the test cases in the same process

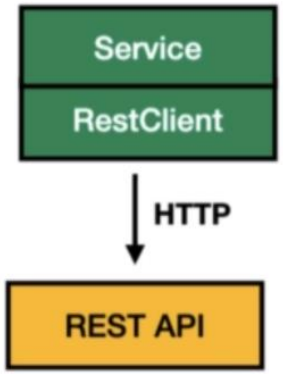
Wiremock

WireMock - StandAlone

- WireMock runs as a separate process

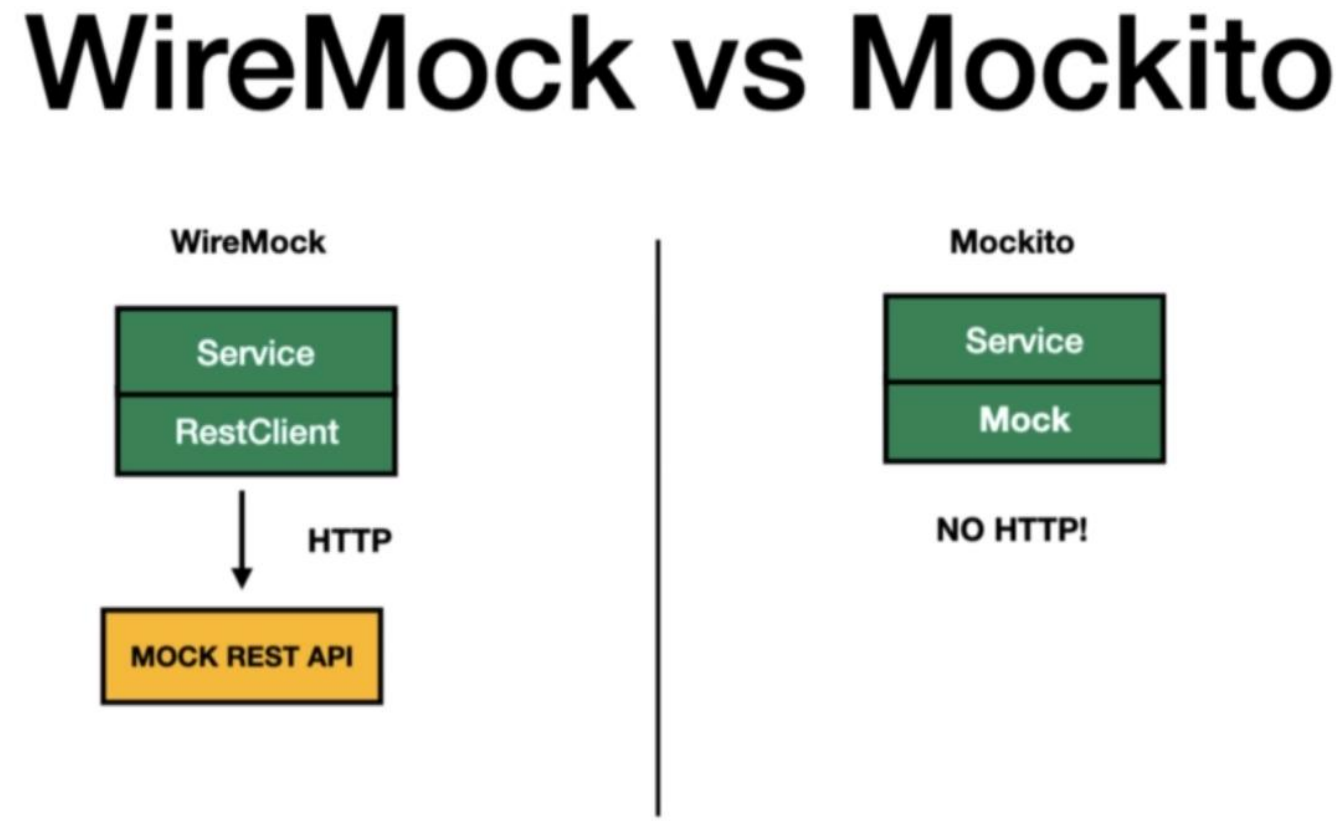
WireMock vs Mockito

- WireMock provides simulation to the actual service
- **Mockito** does the same



Wiremock


Microservices


WireMock vs Mockito
WireMock
Service
RestClient
HTTP
MOCK REST API
Mockito
Service
Mock
NO HTTP!

WireMock vs Mockito

WireMock

- WebServer
- Real HTTP
- Network fault simulation is possible
- Testing equivalent to interacting with production environment

Mockito

- Not a WebServer
- No HTTP
- Network fault simulation is not possible
- Testing not equivalent to interacting with production environment

WebClient vs RestTemplate

WebClient

- Rest API Client
- spring-webflux
- Functional Style API
- Build Synchronous and Asynchronous Rest Clients

RestTemplate

- Rest API Client
- spring-web
- Imperative Style API
- Build only Synchronous Clients

Refer boawk3 customerapiclient

RSQL

- RSQL introduces simple and composite operators which can be used to build basic and complex queries.
- The following table lists basic operators:

```
| Basic Operator | Description         |
|----------------|---------------------|
| ==             | Equal To            |
| !=             | Not Equal To        |
| =gt=           | Greater Than        |
| =ge=           | Greater Or Equal To |
| =lt=           | Less Than           |
| =le=           | Less Or Equal To    |
| =in=           | In                  |
|=out=           | Not in              |
```

RSQL

- These six operators can be used to do all sort of simple queries, for example:
- name==Fero: find all people whose name is Fero
- street!=Darna: find all people who do not live at Darna
- age=gt=10: find all people older than 10 (exclusive)
- age=ge=10: find all people older than 10 (inclusive)
- house=lt=3: find all people who have less than 3 houses
- house=le=3: find all people who have less than or 3 houses

RSQL

The following table lists two joining operators:

```
| Composite Operator | Description         |
|--------------------|---------------------|
| ;                  | Logical AND         |
| ,                  | Logical OR          |
```

RSQL

- These two operators can be used to join the simple queries and build more involved queries which can be as complex as required. Here are some examples:
- age=gt=10;age=lt=20: find all people older than 10 and younger than 20
- age=lt=5,age=gt=30: find all people younger than 5 or older than 30
- age=gt=10;age=lt=20;(str=Fero,str=Hero): find all people older than 10 and younger than 20 and living either at Fero or Hero.

What is AppDynamics and how it can save the day?

- AppDynamics is a leading Application Performance Management (APM) product.
- It is a tool that monitors your Application Infrastructure and gives you code level visibility.
- It is supported for all major technologies (Java, .NET, PHP, Node.js, NOSQL etc) and can be installed either as on-premise or as SaaS (Software As a Service) solution.

What is AppDynamics and how it can save the day?

- A piece of software called Agent is installed in the Application to be monitored.
- The Agent collects the performance metrics and sends them to a Server process called Controller.
- Controller processes the metrics and presents them via Web Browser.
- A monitoring analyst can configure Alerts and generate reports using the Web Interface.

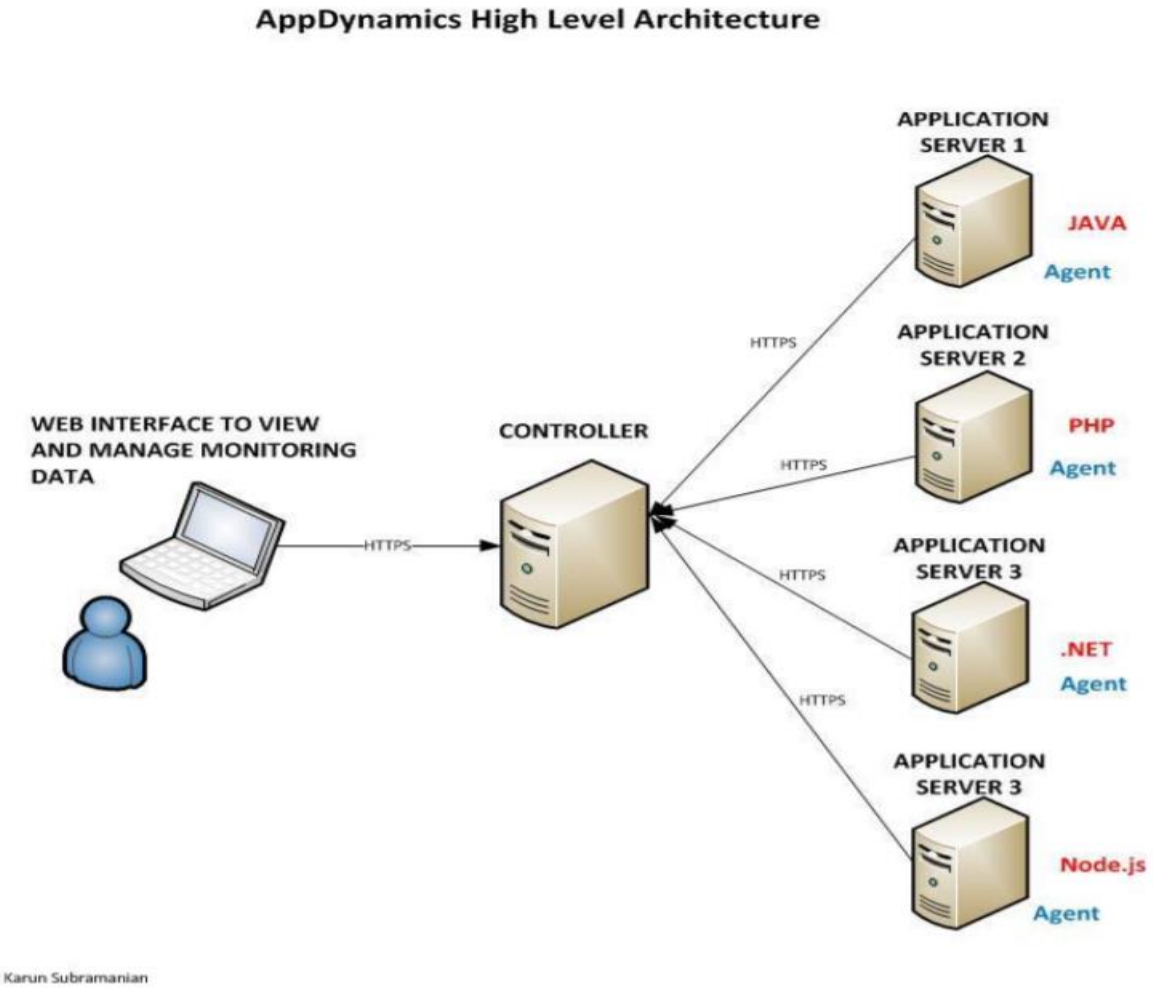
What is AppDynamics and how it can save the day?

- The agent constantly monitors the application.
- Since it uses byte code instrumentation technology, Agent has hooks to every line of code.
- This is how AppDynamics is able to provide code level visibility.
- Agents are available for most of the popular technologies.

What is AppDynamics and how it can save the day?

- Most of the monitoring capability is ‘out of the box’, including and Alerts.
- Another neat ‘out of the box’ feature is the ‘Application Flow Mapping’.
- AppDynamics discovers various subsystems and backends and draws them beautifully on the browser.

Image Source: AppDynamics.com



How can I get started, really?

- 1. Identify a Server that can serve as AppDynamics Controller. Controller is available for the following platforms
  - Windows (32 and 64 bit)
  - Linux (32 and 64 bit)
- Download and install AppDynamics Controller (https://download.AppDynamics.com). Installation is pretty straight forward.

How can I get started, really?

- 1. Identify a Server that can serve as AppDynamics Controller. Controller is available for the following platforms
  - Windows (32 and 64 bit)
  - Linux (32 and 64 bit)
  - MAC
- Download and install AppDynamics Controller (https://download.AppDynamics.com). Installation is pretty straight forward.

How can I get started, really?

- Download and install one of the App agents (Agents are available for the following):
- JVM (Oracle HotSpot, JRockit, IBM JVM)
- .NET
- PHP
- Node.js
- Python
- WebServer on Linux (Apache)
- DB Agent (DB2,Oracle,MySQL,SQL Server, PostgreSQL, Sybase IQ)

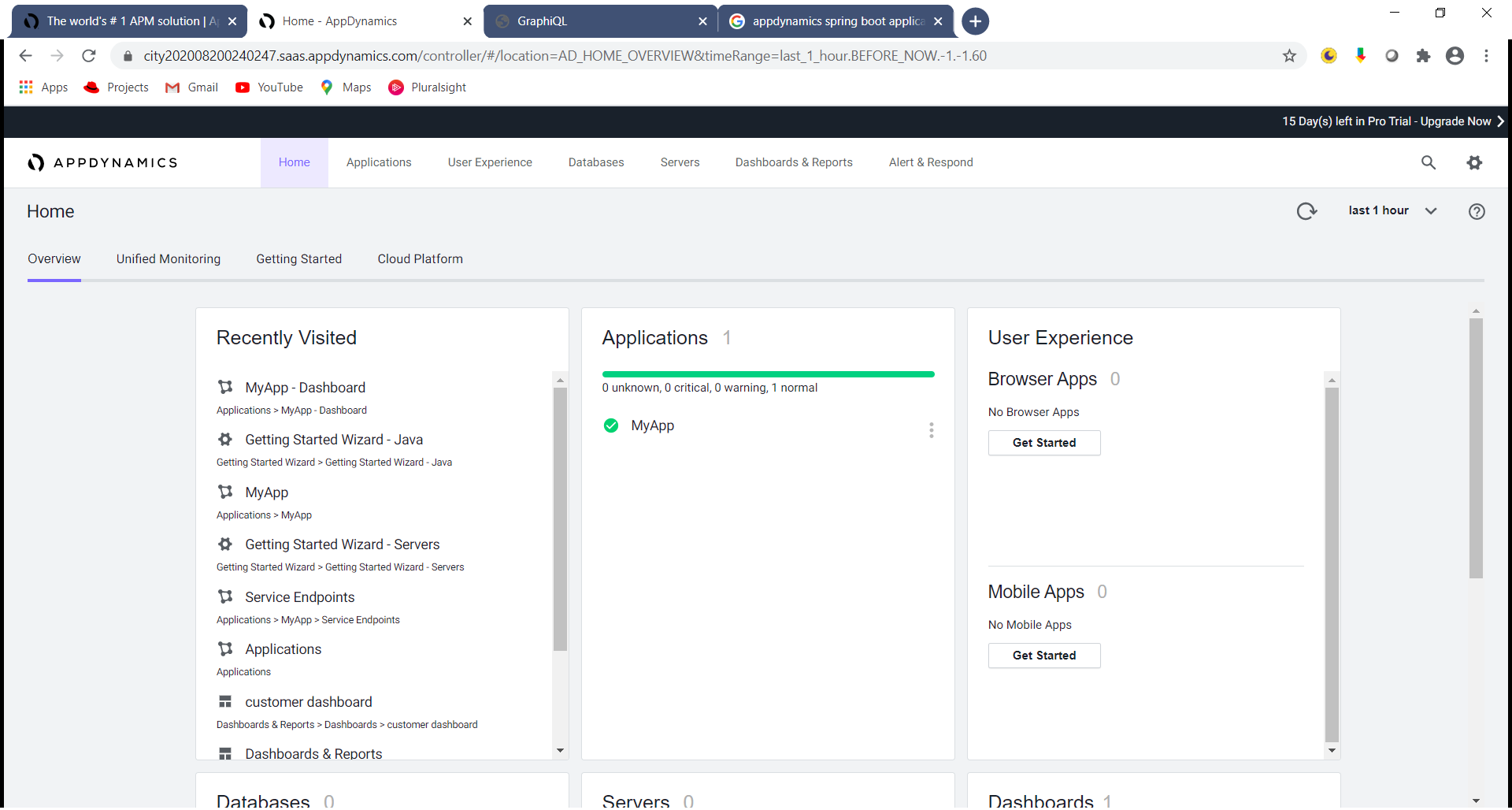
How can I get started, really?

- Configure the App agent; Restart the Application and start monitoring
- The controller listens on port 8090 and 8091. Make sure these ports are not blocked by Firewall.
- Controller on Linux requires libaio installed. Also, the number of File Descriptors available should be at least 65635.

How can I get started, really?

- Configure controller-info.xml
- This file is found at <AGENT_HOME>/conf/controller-info.xml. Here are the important parameters to be configured
- <controller-host>myServer</controller-host>
- <controller-port>8090</controller-port>
- <application-name>AcmeShoppingPortal</application-name>
- <tier-name>OrderFulfillment</tier-name>
- <account-name>***</account-name>
- <account-access-key>***</account-access-key>

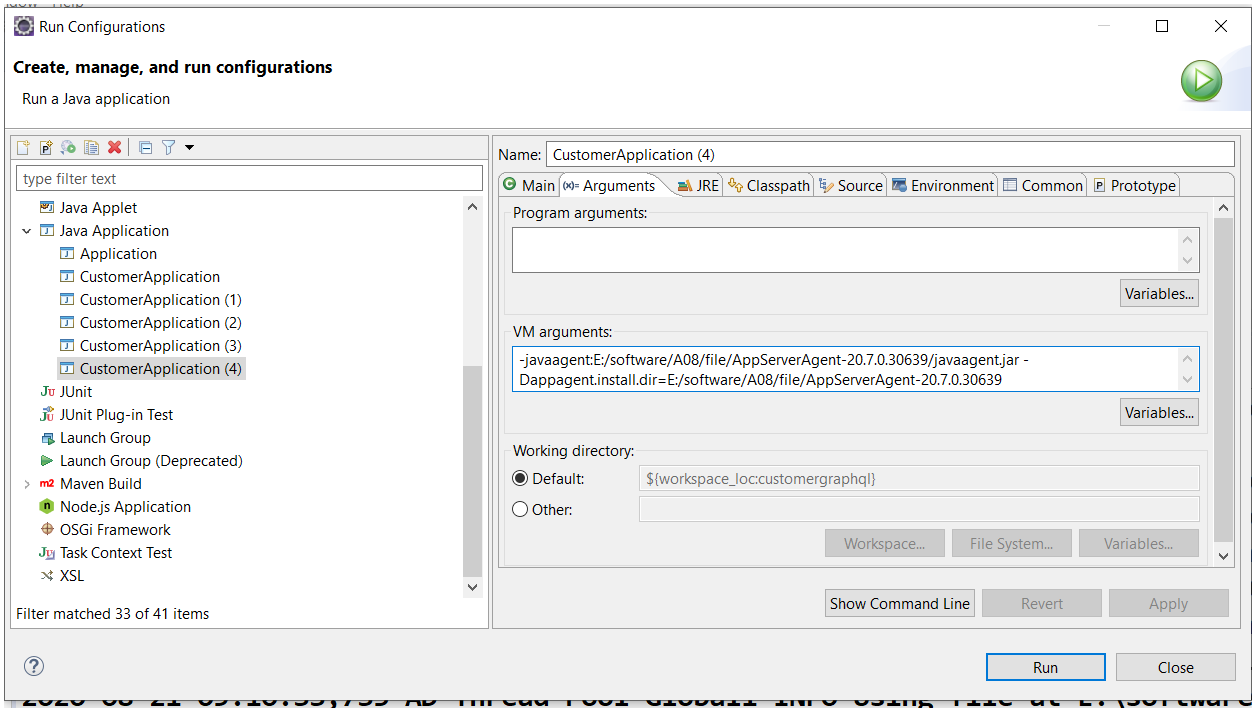
App Dynamics



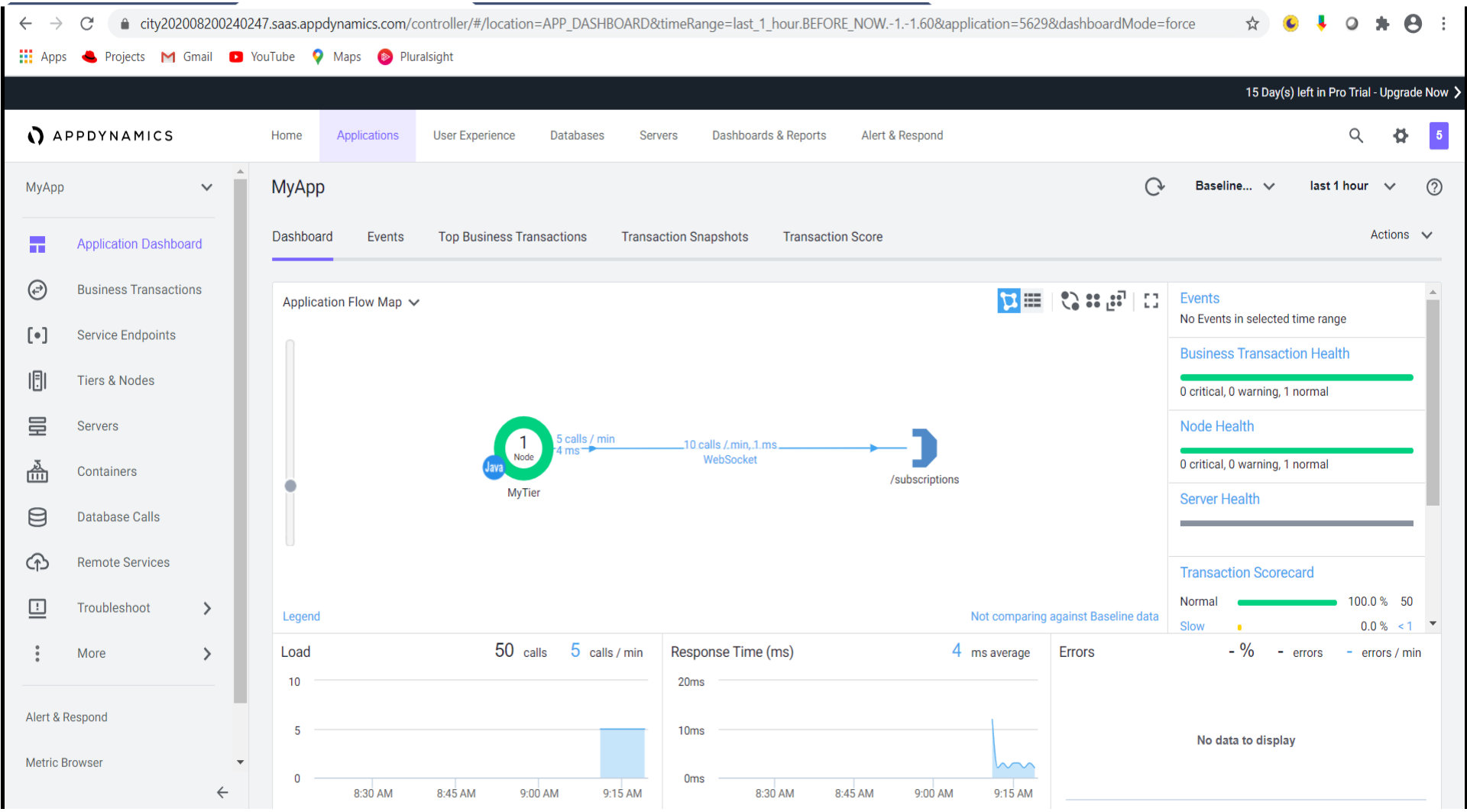
App Dynamics

Microservices

- -javaagent:E:/software/A08/file/AppServerAgent-20.7.0.30639/javaagent.jar -Dappagent.install.dir=E:/software/A08/file/AppServerAgent-20.7.0.30639



App Dynamics



App Dynamics

The world's # 1 APM solution | A
WebSocket./subscriptions - AppD
GraphiQL
Spring Performance Monitoring |
city202008200240247.saas.appdynamics.com/controller/#/location=APP_BT_SLOW_TRANSACTIONS&timeRange=last_1_hour.BEFORE_NOW.-1.-1.60&application=5629&businessTrans...
Apps Projects Gmail YouTube Maps Pluralsight

15 Day(s) left in Pro Trial - Upgrade Now

APPDYNAMICS | Home | Applications | User Experience | Databases | Servers | Dashboards & Reports | Alert & Respond | 5

MyApp

- Application Dashboard
- Business Transactions
- Service Endpoints
- Tiers & Nodes
- Servers
- Containers
- Database Calls
- Remote Services
- Troubleshoot
- More

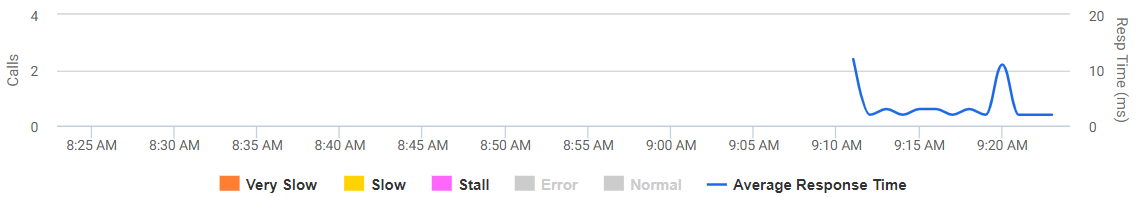
Alert & Respond

Metric Browser

WebSocket./subscriptions

Baseline... | last 1 hour

Dashboard | Events | Slow Response Times | Errors | Transaction Snapshots | Transaction Score | DB Queries



Normal	100.0 %	70
Slow	0.0 %	< 1
Very Slow	0.0 %	< 1
Stall	0.0 %	< 1
Errors	0.0 %	-

Slow Transaction Snapshots

Details | Filters | Analyze | Actions | Configure

Showing 0 of 0

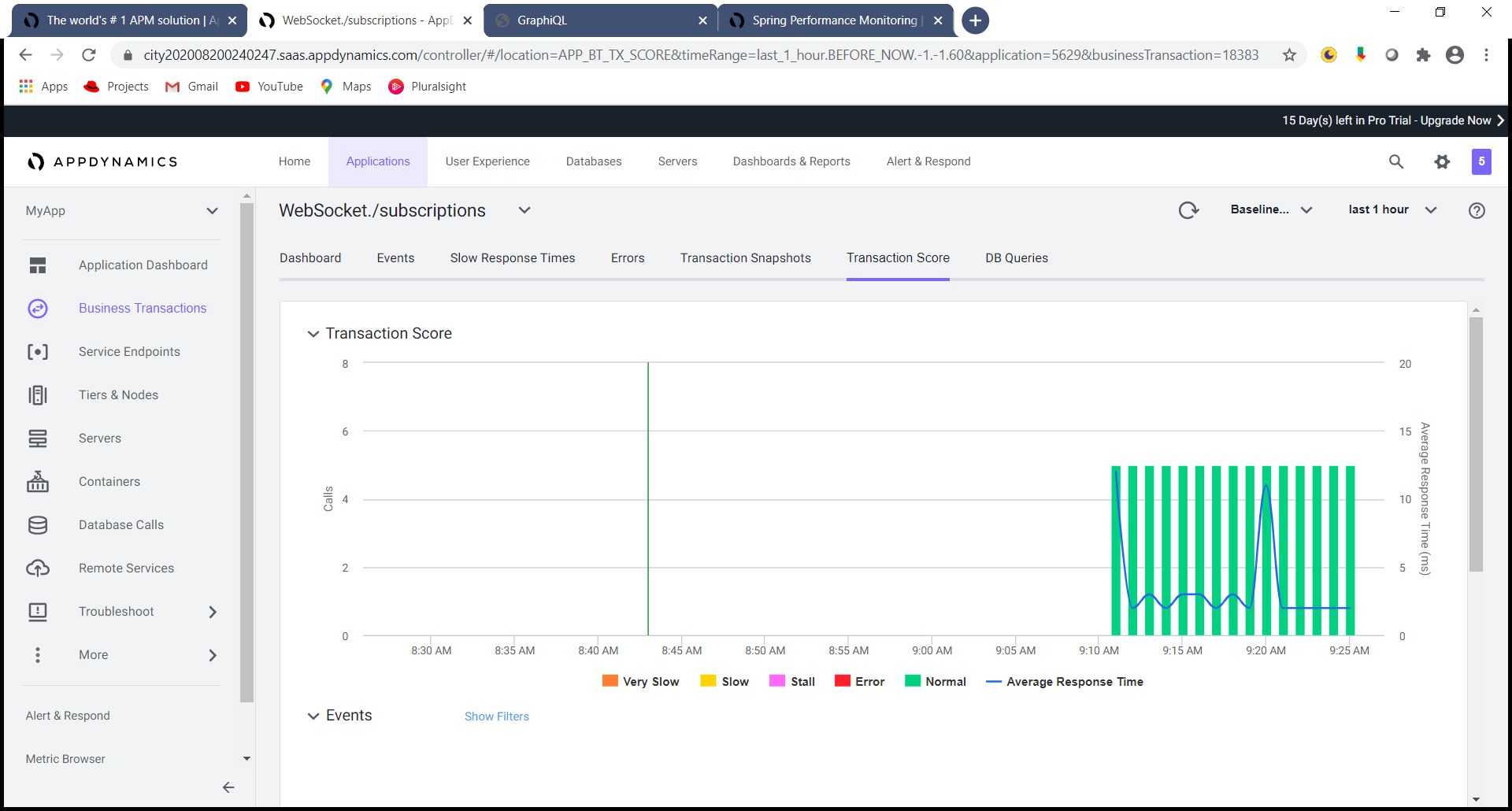
Time	Exe Time (ms)	URL	Business Transaction	Tier	Node

Type here to search

ENG 09:26 21/08/2020

App Dynamics

Microservices




DROPWIZARD

Dropwizard

- Dropwizard is an open source Java framework used for the fast development of RESTful web services.
- It's a light-weight best-in-class set of tools and frameworks for building RESTful web services.

Dropwizard

- Its goal is to provide performant, reliable implementations of everything a production-ready web application needs.
- Because this functionality is extracted into a reusable library, your application remains lean and focused, reducing both time-to-market and maintenance burdens.
- Dropwizard allows a developer to build an initial project very fast - a so-called quick bootstrap project.
- This helps the application to be packaged in a way that allows it to be easily installed on the production environment as a stand-alone service.

Default Dropwizard Components

- Jetty HTTP library
- Jersey
- Jackson
- Metrics: Dropwizard has its own library, that allows us to read the application metrics through HTTP endpoints.
- Guava: is Google's utility library that gives us a large number of classes to speed up development in Java.
- Logback and Slf4j: These two libraries are used for logging, similar to JDK logging (java.util.logging)

Default Dropwizard Components

- Freemarker and Mustache: Choosing a template processor is one of the more important decisions.
- Dropwizard uses well-known and popular and processors to create user interfaces.
- Apache HttpClient: Provides the ability to interact with other web services.
- Hibernate Validator: Used for validating user input.
- Jdbi: Database access classes which have Hibernate support.
- Joda time: Library for handling dates and time.
- Liquidbase: Open source database-independent library for tracking, managing and applying database schema changes.

Mongo Connection

- $ mongo
- $ use donuts
- > db.createUser({ user: "user_donuts", pwd: "pAsw0Rd", roles: [{ role: "readWrite", db: "donuts"}]});
- java -jar target/dropwizard-mongodb-ms-1.0.0-SNAPSHOT.jar server configuration.yml
- http://localhost:7070/dropwizard-mongodb-ms/swagger

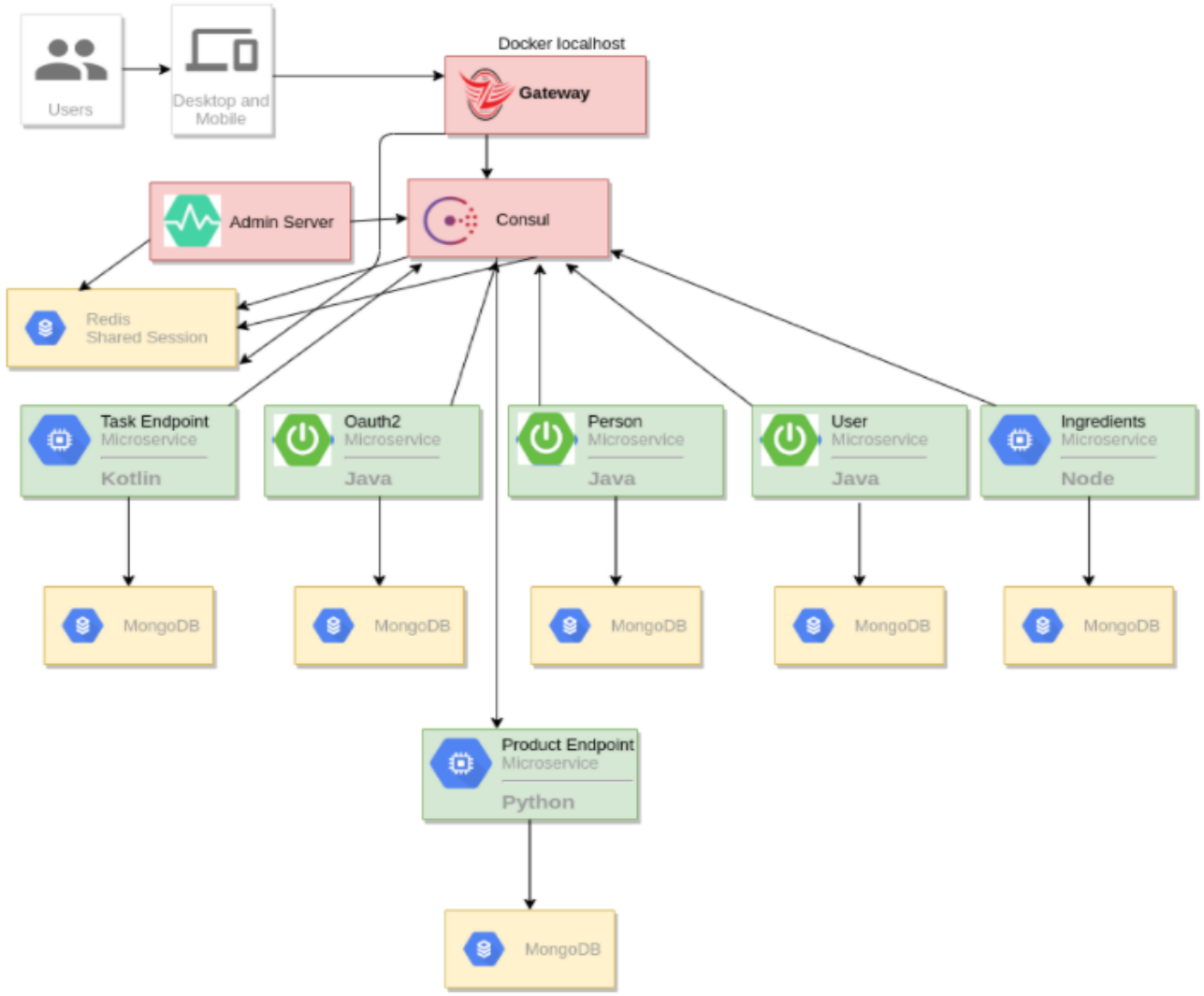
Drop wizard

	Dropwizard	Spring boot
HTTP	Jetty	Tomcat (default), Jetty or Undertow
REST	Jersey	Spring (default), JAX-RS
JSON	Jackson	Jackson, GSON, json-simple
Metrics	Dropwizard Metrics	Spring
Health Checks	Dropwizard	Spring
Logging	Logback, slf4j	Logback, Log4j, Log4j2, slf4j, Apache common-logging
Official integrations	Hibernate Validator, Guava, Apache HttpClient, Jersey client, JDBI, Liquibase, Mustache, Freemarker, Joda time	40+ Official Starter POMs for any purpose
Community integrations	Tens of available integrations, including Spring	4 Community led POMs

Drop wizard

Libraries	Core: Jetty, Jersey, Jackson and Matrics Others: Guava, Liquibase and Joda Time.	Spring, JUnit, Logback, Guava. There are several starter POM files covering various use cases, which can be included in the POM to get started.
Concluding Remarks	If dealing with only REST micro services, drop wizard is an excellent choice.	Where Spring-boot shines is the types of services supported i.e. REST, JMS, Messaging, and Contract First Services. Not least a fully built in Dependency Injection container.

Microservice Diagram



Questions

Module Summary

- Spring Integration Framework.
- Message, Channel and Adapter
- Understood the different Component Integration
- Understood the Event-Driven Architecture