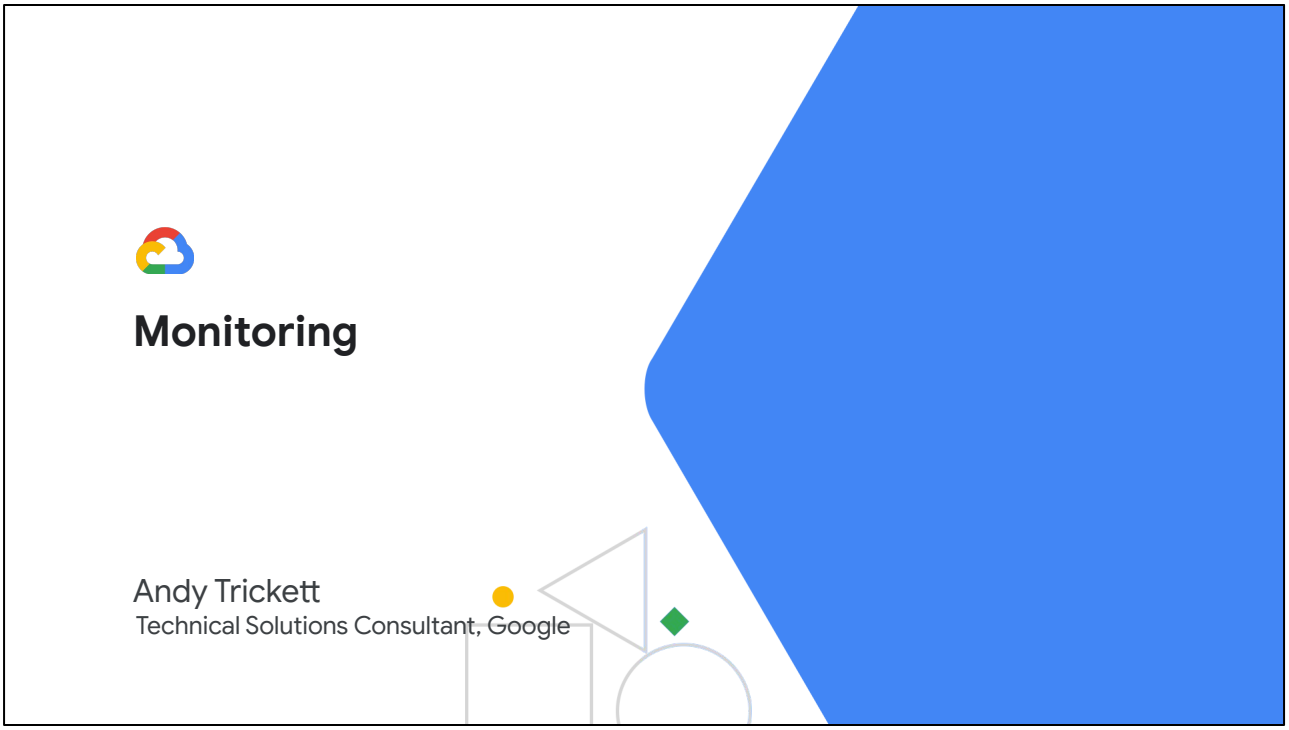


Welcome to the module on monitoring Apigee for private cloud.

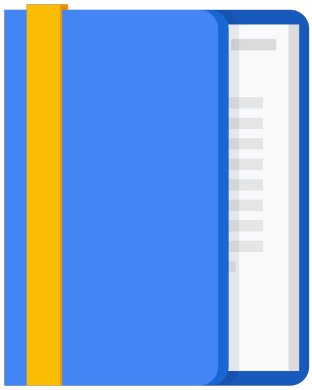
Agenda

Component monitoring

Synthetic transactions

API monitoring

Monitoring strategy



In this module, we discuss the practices used to monitor the components of the Apigee Edge platform for private cloud.

We discuss the role of synthetic transactions as part of a monitoring strategy and describe ways to monitor the API proxies that you deploy to the platform.

And we discuss best practices to establish a scalable monitoring strategy in your organization.

Let’s begin.

System metrics

Here are some of the metrics that are commonly collected for Apigee components and underlying infrastructure:

- **CPU Utilization:** Specifies the basic statistics (User/System/IOwait/Idle) of CPU utilization.
  - An example of this metric is the total CPU used by the system.
- **Free/Used Memory:** Specifies the system memory use in bytes.
  - An example of this metric is the amount of physical memory used by the system.
- **Disk Space Usage:** Specifies file system information based on the current disk usage.
  - An example of this metric is the hard disk space used by the system.
- **Load Average:** Specifies the number of processes waiting to run.
- **Network Statistics:** Network packets and/or bytes transmitted and received, along with the transmission errors about a specified component.

The Apigee private cloud software is a combination of Java, C++, and PHP applications.

As with any software system, basic checks should be put in place to ensure and monitor the health of the operating system, application processes, the network, and hardware.

Items that are frequently monitored include CPU utilization, which accounts for user, system, IO wait, and total CPU, free and used memory, disk space usage, average load on the system, and network statistics.

Thresholds for each of these metrics will vary by component, but it is useful to understand the common patterns for each component.

For instance, Cassandra operates normally most of the time, but during compaction, disk space usage will greatly increase along with the amount of heap memory.

Note that this kind of pattern may be perceived as abnormal for other applications, but is considered normal for Cassandra.

The ideal maximum threshold values will also depend on allocated hardware, network

traffic, TPS throughput, and other factors.

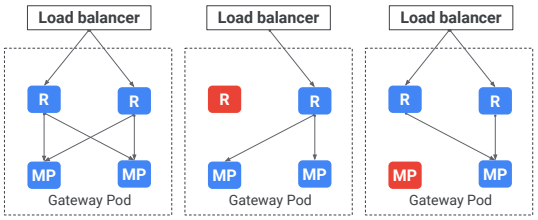
System metrics must be collected for every host and relevant process, and your provisioned infrastructure is expected to provide this capability.

You might find it valuable to view metrics under expected and extreme load conditions. This helps to understand the overall operational patterns to use as input when deciding your thresholds.

Component status

- Edge components expose a service management API on a custom port.
- API calls vary by component, but all components support a set of core calls used to check status:
  - /v1/servers/self
  - /v1/servers/self/up
  - /v1/servers/self/uuid
- To call a component API, use curl or your HTTP client of choice:

```
curl http://router-addr:8081/v1/servers/self/up
```



https://docs.apigee.com/private-cloud/latest/how-monitor#perform-service-checks

Component	Port
edge-management-server	8080
edge-router	8081
edge-message-processor	8082
edge-qpid-server	8083
edge-postgres-server	8084

Validating CPU and memory consumption and the state of the different processes at the operating system level provides relevant information about the health of the server, but it is not sufficient to understand the health of the platform.

Apigee Edge for private cloud provides component-specific APIs for monitoring purposes. These APIs are available for the component services on a specific port.

All components support a core set of APIs, such as /v1/servers/self, which return statistics about the component,

/v1/servers/self/up, which returns true if the component is up, and /v1/servers/self/uuid, which returns the unique identifier, or UUID, of the component.

For example, invoking the /v1/servers/self/up API on a message processor forces the component to execute the API and provide a suitable response.

These self APIs represent a functional test of the component, and their responses are considered successful when the API responds with a HTTP 200 status code that indicates success.

A full set of APIs for each component is at the link provided.

Log events

Router health check of the Message Processor

Mark Down event

```
2021-06-07 06:18:51,589  RPCClientClientProtocolChildGroup-RPC-0 INFO  CLUSTER -
ServerState.setState() : State of 7b4b7bac-2104-417d-821a-361690380761 is now HB_FAILED.
handle = <MP_IP> at 1623046731589

2021-06-07 06:18:52,956  RPCClientClientProtocolChildGroup-RPC-0 INFO  CLUSTER -
ServerState.setState() : State of 7b4b7bac-2104-417d-821a-361690380761 is now DISCONNECTED.
handle = <MP_IP> at 1623046732956
```

Mark Up event

```
2021-06-07 06:17:21,465  RPCClientClientProtocolChildGroup-RPC-0 INFO  CLUSTER -
ServerState.setState() : State of 7b4b7bac-2104-417d-821a-361690380 761 is now STARTING.
handle = <MP_IP> at 1623046641465

2021-06-07 06:07:46,014  RPCClientClientProtocolChildGroup-RPC-0 INFO  CLUSTER -
ServerState.setState() : State of 609a6bab-746b-4721-9cce-be79c3eba369 is now CONNECTED.
handle = <MP_IP> at 1623046066014
```

All the Apigee Edge components log messages to disk. Log files reside in the /opt/apigee/var/log component directory.

When logging at the default logging level, system logs contain exceptions, errors, and health check activity.

For example, the router component is designed to initiate message processor health checks.

If the message processor is marked as down or unreachable, the router will take the message processor out of rotation until the health check starts to pass again.

These events are observed in the router log file as markdown and markup events and captured in the /opt/apigee/var/log/edge-router/logs/system.log file.

Metrics

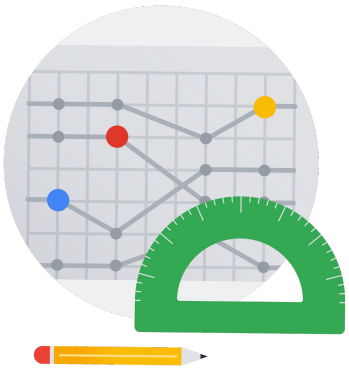
Apigee gathers run-time metrics on router and message processors.

Metrics include details about:

- All outbound traffic to a given host IP address
- All inbound traffic for a given organization
- Many others

Run-time metrics are collected in three groups:

- Cumulative: collected from server start
- Current: collected in the last minute
- Previous: copied from current every minute



Apigee Edge enables you to collect component run-time metrics. Edge starts with all metrics in all groups set to zero.

Cumulative metrics contain the values collected from the start of the server.

The current metrics group contains metrics collected in the last minute.

Every minute, the metrics from the current group are copied to the previous metrics group, and the metrics in the current group are reset to 0.

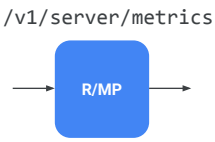
Metrics API

Get the complete metrics object:

```
curl -v http://<RMP_IP>:<port>/v1/server/metrics
```

Get details about all inbound traffic:

```
curl -v http://<RMP_IP>:<port>/v1/server/metrics/inbound/traffic
```



Get inbound traffic for a specific organization:

```
curl -v http://<RMP_IP>:<port>/v1/server/metrics/inbound/traffic/organizations/<org_name>
```

Get inbound traffic for a specific environment:

```
curl -v
http://<RMP_IP>:<port>/v1/server/metrics/inbound/traffic/organizations/<org_name>/environment
s/<env_name>
```

Get details about all outbound traffic:

```
curl -v http://<RMP_IP>:<port>/v1/server/metrics/outbound/traffic
```

Get outbound traffic for a specific host:

```
curl -v http://<RMP_IP>:<port>/v1/server/metrics/outbound/traffic/hosts/<host>
```

You can get a complete list of metrics by using the /server/metrics API for the Edge router and message processor components.

The output is in JSON format and contains all metrics: cumulative, current, and past.

You can also filter the metrics by specifying a resource path.

For example, to get metrics on all inbound traffic, you can specify the /server/metrics/inbound/traffic path.

To get traffic for a specific organization or environment, you can further filter the request by adding the organization and environment names to the path.

JMX monitoring

- Apigee Java components expose JMX interfaces for you to consume. The Mbeans that are exposed only provide read operations:

  ```
  service:jmx:rmi:///jndi/rmi://<ip address>:<port>/platform
  ```

- By default, these interfaces are not protected by authentication. JMX authentication can be enabled using:

  ```
  /opt/apigee/apigee-service/bin/apigee-service edge-management-server
  change_jmx_auth -f <configFile>
  ```

  Additional details are provided here: docs.apigee.com/private-cloud/latest/how-monitor#jmx-auth

- Open source components such as Cassandra also expose JMX interfaces:

  ```
  service:jmx:rmi:///jndi/rmi://<ip address>:7199/jmxrmi
  ```

  Additional details are provided here:
  docs.apigee.com/private-cloud/latest/how-monitor#cassandra-jmxstats

Apigee Edge enables you to collect component runtime metrics via JMX. These metrics are exposed using an Mbean, called platform.

On this JMX interface, metrics related to inbound and outbound traffic, thread pools, memory, and others are available.

Authentication is not enabled by default on the Edge JMX Mbean, but it can be enabled if required.

Many open source components, such as Cassandra, Qpid, and ZooKeeper, also provide JMX interfaces which can also be used.

Detailed documentation on the Apigee Edge JMX interfaces is at the links provided.

Apache ZooKeeper

ZooKeeper status can be verified by:

- Using apigee-service:

  ```
  /opt/apigee/apigee-service/bin/apigee-service apigee-zookeeper status
  ```

- Checking whether the PID file exists:

  ```
  /opt/apigee/var/run/apigee-zookeeper/apigee-zookeeper.pid
  ```

- Checking whether the ports are open:

  ```
  2181/TCP 3888/TCP
  ```

You can also run the four-letter command `ruok` to test whether the server is running:

```
echo ruok | nc <host> 2181
```

A successful response returns `imok`.

You can monitor ZooKeeper by confirming whether the process is up and running.

You can do this using the apigee-service utility or by checking for open ports and the existence of the ZooKeeper PID file.

Another option is to test whether the service responds to a client request by sending a four-letter command to the ZooKeeper service port: 2181.

For example, if you send the command “ruok” (Are You OK), you should get a response: “imok” (I Am OK) if ZooKeeper is up and running.

Apache Cassandra

- To monitor Cassandra, run the command:

  ```
  /opt/apigee/apigee-cassandra/bin/nodetool -h localhost ring
  ```

  Look for the columns Status: `Up` and State: `Normal` in the output for all nodes.

```
Datacenter: dc-1
==========
Address      Rack Status State   Load    Owns    Token
                                                  113427455640312821154458202477256070485
10.5.10.10   ra-1 Up     Normal  3.24 GB 100.00% 0
10.5.10.20   ra-2 Up     Normal  3.84 GB 100.00% 56713727820156410577229101238628035242
10.5.10.30   ra-3 Up     Normal  3.55 GB 100.00% 113427455640312821154458202477256070485
```

- You can also check thrift status:

  ```
  nodetool -h localhost statusthrift
  running
  ```

In addition to using JMX, Cassandra can be monitored using its management utility nodetool.

The nodetool ring command gives you the status of each node that is part of the Cassandra ring.

You can parse the output of the command to search for the node's status (Up) and state (Normal).

Cassandra uses the thrift protocol to communicate with its clients.

You can check whether Cassandra is listening for client connections by running the nodetool statusthrift command.

The expected result is the string "running".

Openldap

Use ldapsearch (yum install openldap-clients) to query the entry of the system admin.

This entry is used to authenticate all API calls.

```
ldapsearch -b "uid=admin,ou=users,ou=global,dc=apigee,dc=com" -x -W -D
"cn=manager,dc=apigee,dc=com" -H ldap://localhost:10389 -LLL
```

You can monitor the Openldap component to confirm that specific LDAP search requests return the correct results.

The sample LDAP search request will query the entry for the sysadmin user.

Monitoring summary spreadsheet

Refer to the summary document, which contains details on the methods that you can use to monitor Apigee Edge from different perspectives:

- Network
- Hardware
- Operating system
- Application component
- API Proxy
- JMX

A monitoring spreadsheet that contains details of methods that can be used to monitor Edge components from different perspectives is available at the link provided.

The monitoring perspectives used are:

Network: you can either use the management APIs embedded in each Edge component or use other tools to validate whether the components are listening on their dedicated ports.

Hardware: If you've installed Apigee on bare-metal hardware, additional items can be monitored, such as CPU, disks, and NICs.

Operating system: From the OS perspective, you can monitor hardware resource utilization for CPU, RAM, Hard disk, IO, and Network. You can also monitor these resources while the components processes are running.

Application: Monitoring checks from the component application perspective is at a higher abstraction level. Several methods to check on the health of the application include directly querying the application with helper scripts and tools.

API Proxy: You can embed log collection and custom scripts in your API proxy code to help you determine the health and performance of your API transactions. You can also use "synthetic" transactions, which are discussed later in this module.

JMX attributes: You can also monitor internal JVM and Cassandra attributes via the JMX console. A list of these attributes is included in the monitoring summary document.

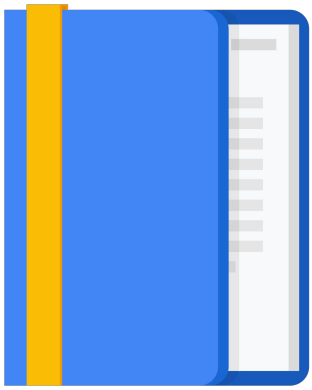
Agenda

Component monitoring

Synthetic transactions

API monitoring

Monitoring strategy



Monitoring individual components is required but not sufficient.

The fact that each component may be up and running does not guarantee that the system is working.

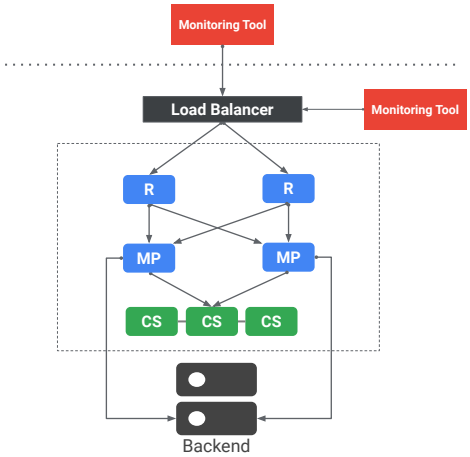
In this lesson, we discuss how to leverage synthetic transactions to enhance your monitoring capabilities.

Synthetic transactions (1/4)

Synthetic transactions are used to verify the functional health of components on the critical path.

This is achieved by implementing an API Proxy with a set of basic operations that is used as a heartbeat.

Here is a basic implementation of the Heartbeat API Proxy.



A synthetic transaction is a lightweight component that is used to test connectivity between elements but does not change the state of any data or systems along the test path.

Routers, message processors, and Cassandra are the Apigee components that are on the critical path in an API transaction.

For the purposes of this course, we consider a synthetic transaction to be an action that is executed over a resource with the goal of validating system runtime availability.

Your monitoring strategy should consider executing synthetic transactions from both within your network and outside your network.

The core functionality of Apigee Edge can be leveraged for synthetic transaction monitoring using an API proxy that executes specific actions on the system.

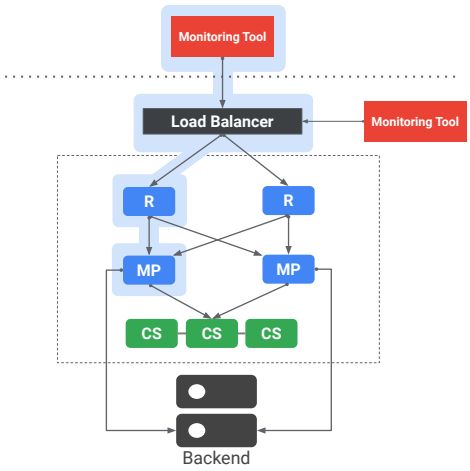
The heartbeat API proxy is one such implementation. You can download the heartbeat proxy from the training material or resources section of the course.

Synthetic transactions (2/4)

A no target operation.

The request is resolved on the Message Processor without any invocation to Cassandra.

```
/v1/heartbeat/rmp
```



The sample heartbeat proxy implements a set of API operations or resources that can be used for synthetic transaction monitoring.

The proxy contains the /rmp resource, which implements an API flow that executes actions over the router and message processor.

The resource path is configured as a “no target” operation, so the response is being provided by the message processor.

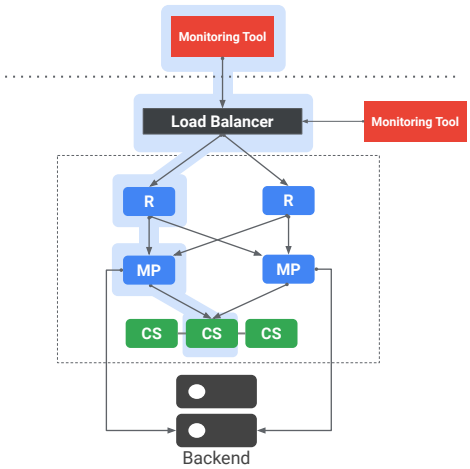
This approach allows us to monitor the availability and latency of Apigee routers and message processors without considering the Cassandra component or backend systems.

Synthetic transactions (3/4)

A no target operation.

The request is resolved on the Message Processors while invoking Cassandra to calculate Quota.

`/v1/heartbeat/rmpcs`



The /rmpcs resource of the heartbeat proxy implements an API flow that executes actions over the router, message processor, and Cassandra components.

This resource path is implemented using the Apigee quota policy that involves the message processor component accessing the Cassandra database.

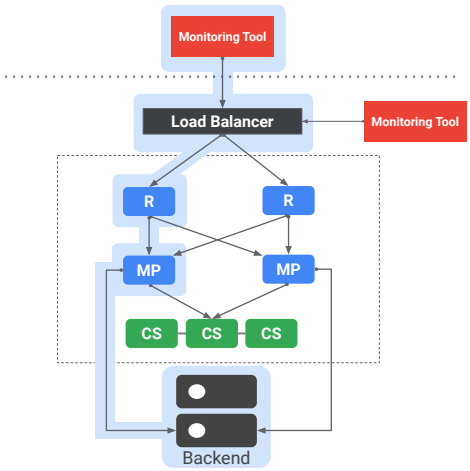
To get a successful response from this API resource, all of the router, message processor, and Cassandra components need to be up and running.

Synthetic transactions (4/4)

A passthrough operation.

The request will result in a target server invocation. The operation returns the response from the backend system.

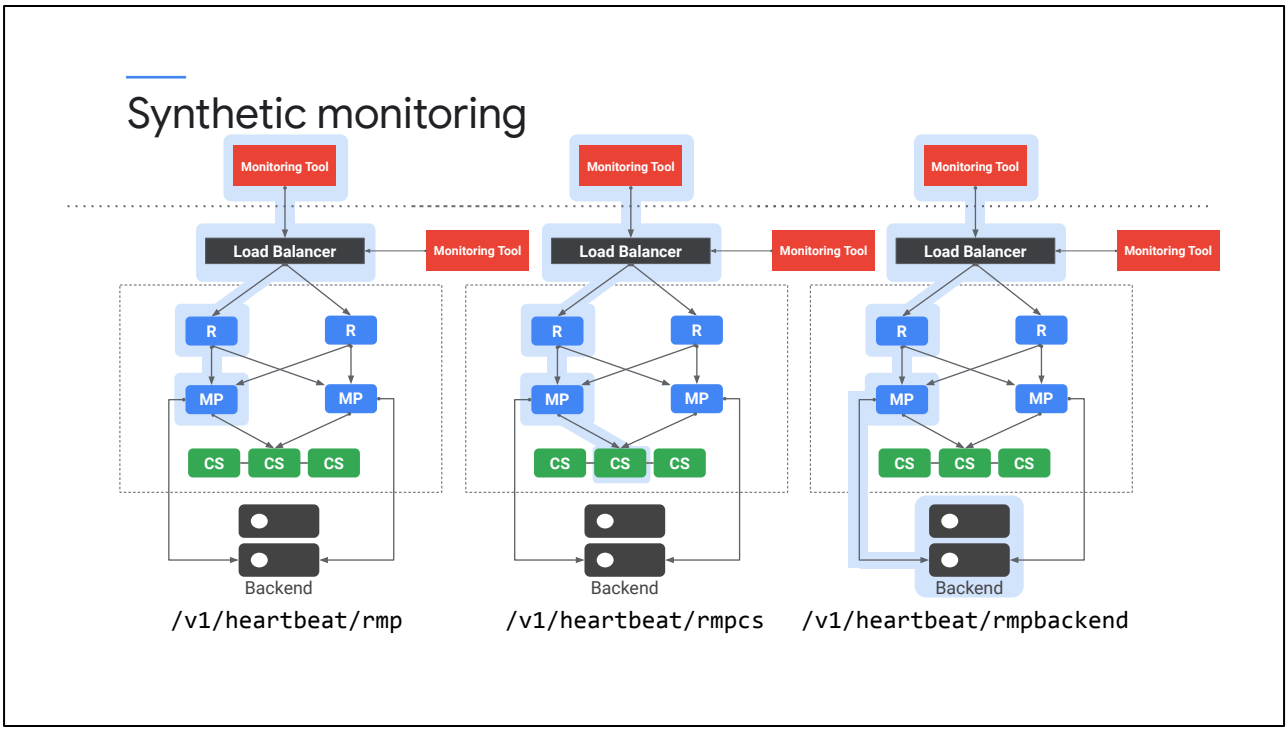
`/v1/heartbeat/rmpbackend`



The /rmpbackend resource of the heartbeat proxy implements an API flow that executes actions over the router and message processor before invoking the backend system.

The response from the API resource will be that returned from the backend system.

Note that the objective here is not to test every API or service offered by the backend system, but instead to validate the message processor's ability to connect to the backend.



Synthetic transaction monitoring can be used to complement component monitoring.

It provides increased visibility and helps in issue identification.

By using these and other checks, you can quickly identify issues with the platform.

The heartbeat API proxy is available in the linked resources and training materials for this course. You can customize it for your requirements.

Heartbeat API proxy

Configuration and deployment steps:

1. Import the heartbeat-api-proxy.zip to the organization.
2. Update the definition to match available virtual hosts in your environment.
3. Create a target server with the following name: HeartBeatTarget.
   - The proxy assumes that the backend service supports the /rmpbackend resource.
   - You can implement this resource on the backend service or modify the proxy to use an existing backend resource.
   - To modify the proxy, refer to this community article.
4. Modify the heartbeat-api-proxy to change its behavior and customize it for your needs.
5. Deploy the heartbeat-api-proxy to your environment.

To use the heartbeat API proxy, you will need to import it to your organization and configure it for your deployment.

To import the proxy zip file to your organization, use the Apigee UI or management API.

Then use the editor in the UI to update the name of the virtual host in the proxy configuration to match the name of the virtual host defined in your environment.

Next, create a target server definition with the name HeartBeatTarget and configure it to point to your backend service. The target server should point to any available HTTP resources in the backend system.

Note that the goal of this test is to monitor the connection to the backend system from the Edge message processor component.

By default, the API proxy will attempt to call a resource on the backend system located at /rmpbackend. If you cannot create this resource on the backend system, you can change the target path in the API proxy configuration.

A good example of how to do this is on the Apigee community website at the link provided.

You can further customize the heartbeat proxy for your requirements.

Heartbeat API proxy

Usage

Check the Router and Message Processor:

```
curl -v http://<host>/v1/heartbeat/rmp
```

Check the Router, Message Processor, and Cassandra:

```
curl -v http://<host>/v1/heartbeat/rmpcs
```

Check the Router, Message Processor, and Backend target endpoint:

```
curl -v http://<host>/v1/heartbeat/rmpbackend
```

Expected success response: **`HTTP/1.1 200 OK`**

The heartbeat API can be invoked using the supported API resource paths.

These calls should be automated and executed from within your monitoring infrastructure.

To check for success or failure, the automation scripts can validate the HTTP status code response from the APIs.

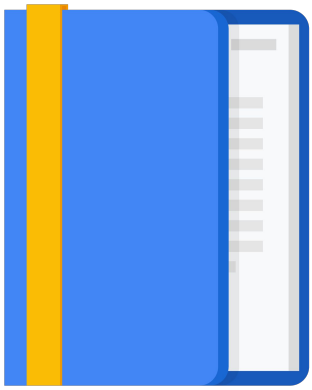
Agenda

Component monitoring

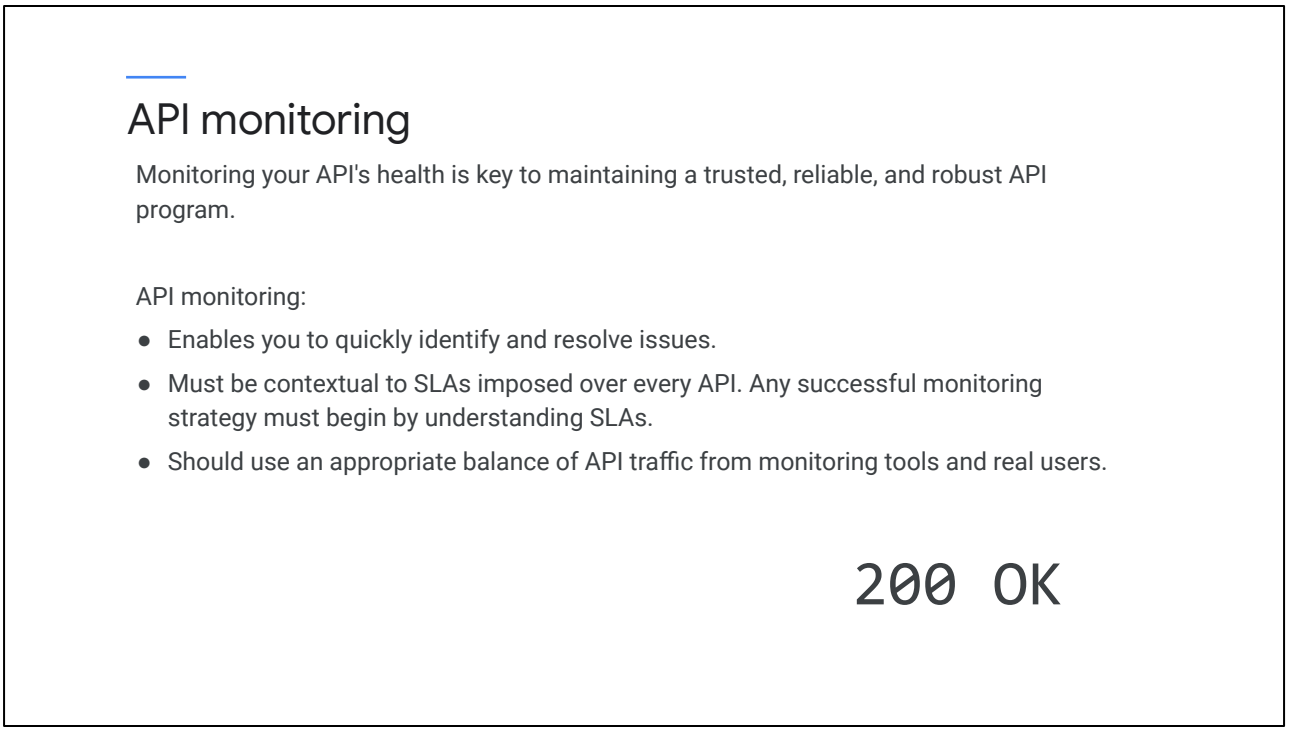
Synthetic transactions

API monitoring

Monitoring strategy



Let’s discuss some of the approaches that you can use to monitor API proxies that are deployed on the Apigee private cloud platform.



Monitoring the health of your APIs is key to a reliable and successful API program and is valuable for quickly identifying and resolving issues.

We've already covered component and synthetic transaction monitoring in previous lessons.

Although those types of monitoring are vital, ultimately the function of an API platform is to service API requests within a defined service level agreement, also known as an SLA.

Any successful monitoring strategy begins with understanding your SLA.

When designing your API, consider how it can be monitored in a lightweight and maintainable fashion.

API monitoring strategy

Consider the following questions:

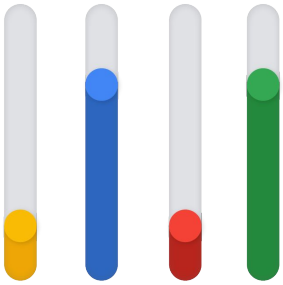
What are the requirements for monitoring API health?

- Is a simple status check enough?
- Are certain resources critical?
- How deep must the monitoring be?

Are there requirements for monitoring the health of the target endpoint through the API?

Which environments are important to monitor?

- Production
- Integration
- Other?



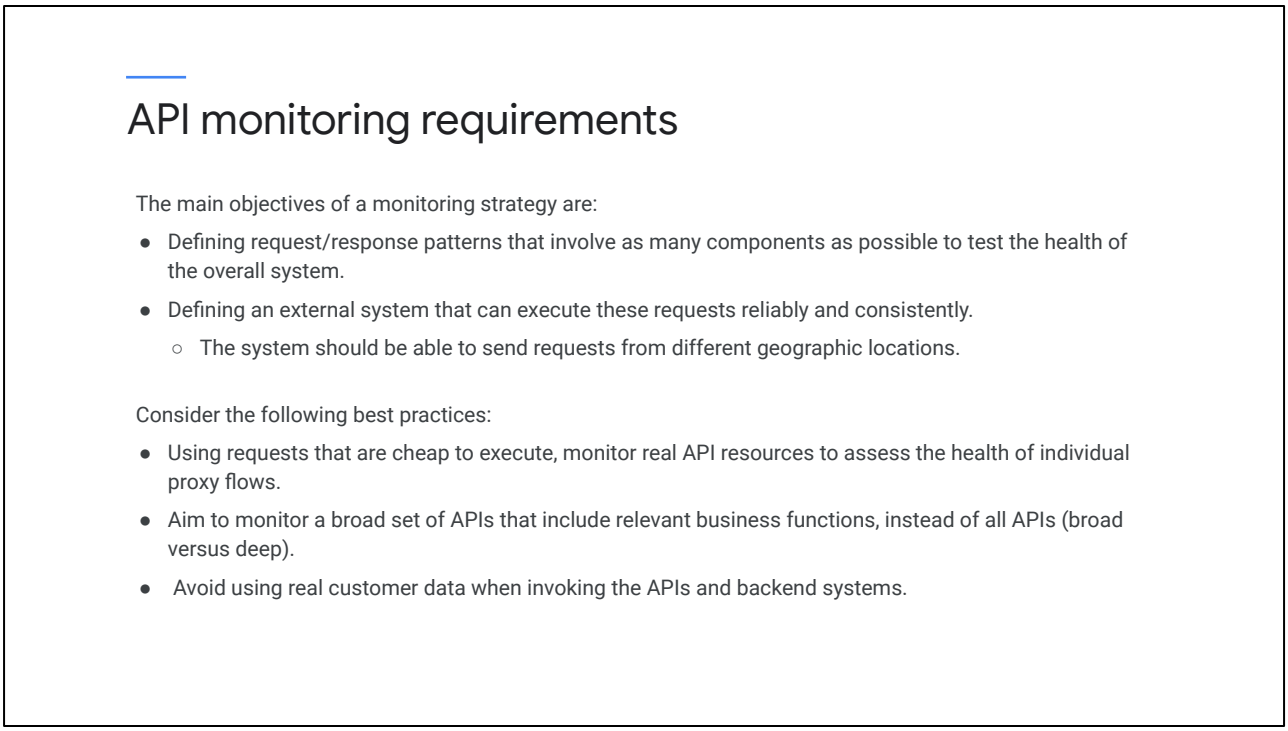
Differentiating between API proxy health versus the health of the target backend can be key when diagnosing issues in production and non-production environments.

When you begin thinking about how to monitor your APIs, you should ask probing questions about the depth of monitoring that is required to meet your service level agreements.

Is it enough to know that a service is up? Or, must you also be able to validate that all critical resources are available?

Can you monitor only the API endpoint proxy to meet your SLA, or should you also be monitoring the target endpoint?

While it is obviously necessary to monitor production services, it is often important to monitor beta, development, and integration environments, because they can be mission critical to other groups.



The main objectives of a successful monitoring strategy include defining requests and response patterns that involve as many components as possible, and defining an external system or systems that can execute these tests consistently.

Instead of targeting every API proxy, prioritize a subset of proxies by relevant business function that invoke every backend system that is consumed by your APIs.

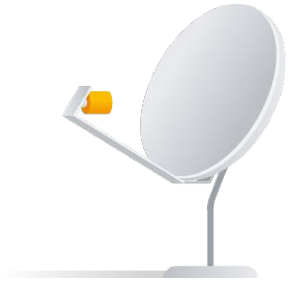
The goal is to ensure that all of the components function together to successfully service requests within the SLA.

A general best practice is to identify small inexpensive requests to real API resources that use multiple components. In this context, broad coverage is more valuable than deep coverage.

Also when making monitoring requests to the APIs and backend systems, avoid the use and exposure of real customer data.

Common patterns

- Ping sub resource
  A specialized sub resource exposed by the proxy to:
  - Test proxy network connectivity
  - Test proxy deployment status
- Target resource
  A specialized resource to:
  - Test proxy-to-target network connectivity
  - Assess target API health
- Real resource
  Use existing API resources to check the health of the system.



Three common patterns are found in successful monitoring programs.

First, a specialized sub-resource is used to test network connectivity and API proxy deployment status.

Second, a specialized target resource is used to report status of the backend systems.

And third, use real API requests to known resources with known states to check the health of the system and its ability to return a valid response.

Using Apigee Analytics for API monitoring

Use Apigee Analytics to monitor APIs:

- Create a separate monitoring client app.
- Use requests to real API resources.
- Identify requests in analytics graphs.
- Include identifying information in the request.
- Use this information to filter out monitoring requests in analytics reports.

If your API is protected by API keys, OAuth, or some other authentication scheme, consider creating an application solely for the purpose of monitoring.

Using a separate monitoring application lets you exclude monitoring calls from your analytics reports and dashboards that plot real traffic to the platform.

It also helps to easily check on the health of the platform using monitoring traffic from the application.

API monitoring tools

Several tools can help you monitor your API.
Here are some commonly used tools:

- Google Cloud Operations
  - cloud.google.com/products/operations
  - cloud.google.com/monitoring/uptime-checks
- Librato: www.librato.com
- Pingdom: www.pingdom.com
- Runscope: www.runscope.com
- API Metrics: apimetrics.io
- Uptime: www.redotheweb.com/uptime

Several tools can help you monitor your API program.

Google offers Cloud Operations suite (formerly called Stackdriver), which is a Google cloud offering that helps to aggregate logs, monitor services, and create alerts.

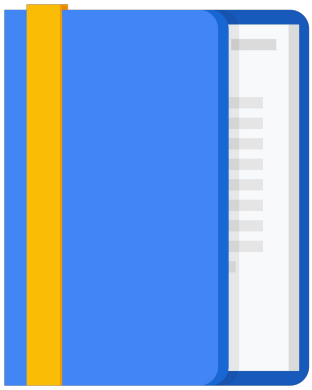
Agenda

Component monitoring

Synthetic transactions

API monitoring

Monitoring strategy



Let’s briefly discuss some principles that can be used when creating a monitoring strategy for your organization.

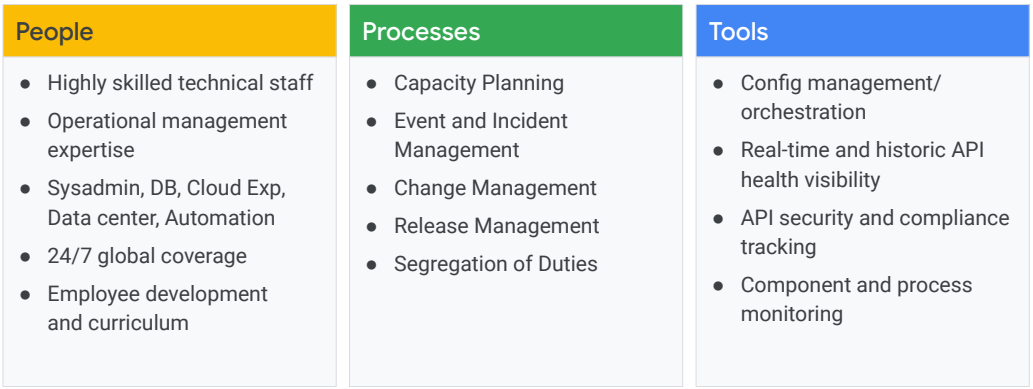
What are we trying to answer?

What..., When..., Who..., How..., Why...

The goal of a sound monitoring strategy is to be able to fully answer questions about the events that occur during platform operation.

In this lesson, we discuss some of the practices that can be used to achieve this goal.

The 3 pillars



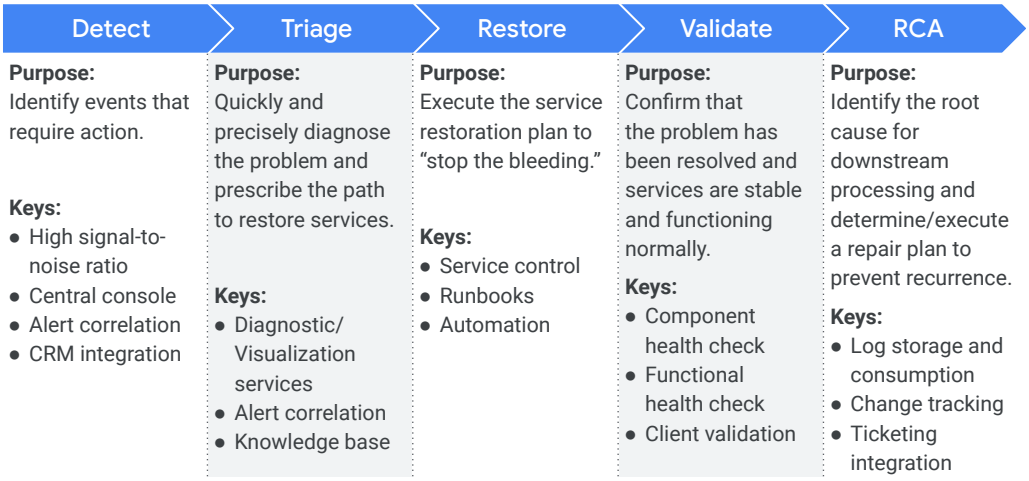
A well-defined monitoring strategy relies on trained personnel, well-documented processes, and having the right tools in place.

Personnel are typically system administrators with experience in cloud operations, database management, devOps automation, and general operations.

Processes to manage your Apigee private cloud operations should include event and incident management, change and release management, and capacity planning.

To support your operations tasks, you need tools for configuration management, system observability and monitoring, and security and compliance tracking.

Support incident workflow



As part of your monitoring and troubleshooting process, adopting a well-defined support and incident workflow is critical.

The process should cover various phases in troubleshooting an issue, from detection, triage, service restoration, and validation, to root cause analysis.

Each phase incorporates distinct capabilities.

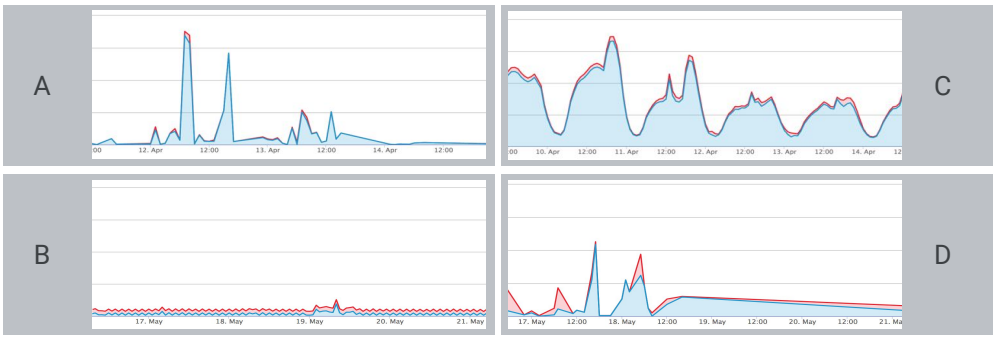
It is important to have a central tool or console to detect and triage issues with the platform. A well-designed tool will have alert generation and integration with CRM tools to manage customer reporting and also visualization and diagnostic capabilities.

Using runbooks and automation is a repeatable best practice when restoring services. Component and functional health checks are essential pieces of a service validation strategy after the service has been restored.

A root cause analysis must always be performed for service outages in production environments. It helps document the cause and effect of the problem and helps to implement procedures to prevent recurrence.

Cloud scale and multi-tenancy

- Large number of customers
- Large number of APIs per customer
- Multi-tenant infrastructure
- Distinct domain and business rhythm
- Distinct API traffic patterns



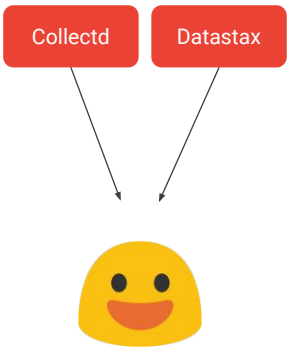
Apigee Edge for private cloud is built with multi-tenancy as a core design principle.

You might host multiple tenants, each with one or more environments on the platform.

You can use the built-in analytics features of the product to gather intelligence for each and every one of those environments.

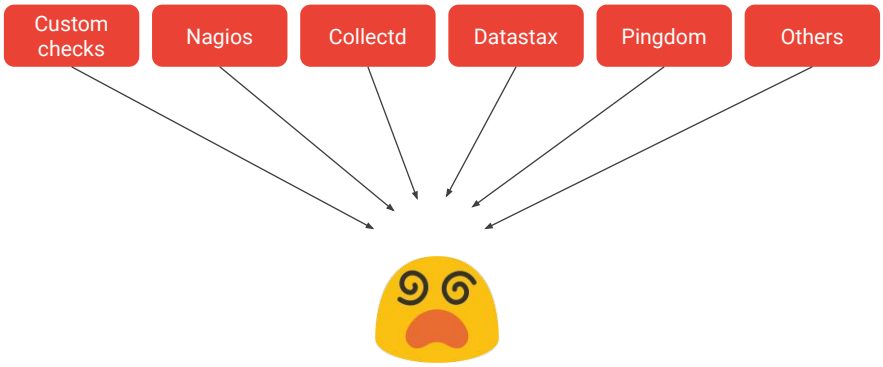
It is your responsibility to analyze and understand the needs and traffic patterns of your tenants, because they will often have different monitoring and support needs.

Traditional information silos



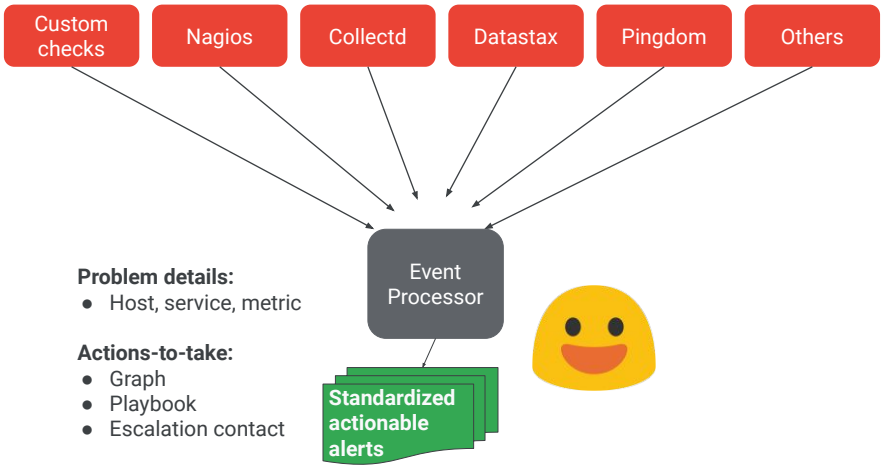
Having insights on each and every component of the system is important and improves the observability and maintainability of the system.

Traditional information silos



But if those metrics are collected by many different tools with different notification and alert channels, they can easily overwhelm the monitoring and operations teams.

Traditional information silos



A good practice is to build or install an event processing system that collects the inputs from the various monitoring tools and standardizes the resulting message format, delivery, and alerting process.

This helps with the next stage of the monitoring process, which is to create actionable alerts and trigger playbooks.

Standardized actionable alerts and automation

Creating automated actionable alerts helps in the rapid resolution of problems.

Actionable alert	Playbooks	Automation	People
• Service • Tags • Metric • Trigger • Host • Playbook	• Problem characteristic • Triggers • Resolution steps • Escalation (to another playbook or human)	Automatic execution of playbooks based on triggers	• Focus on complex problems. • Focus on actionable alerts, playbook, and automation.

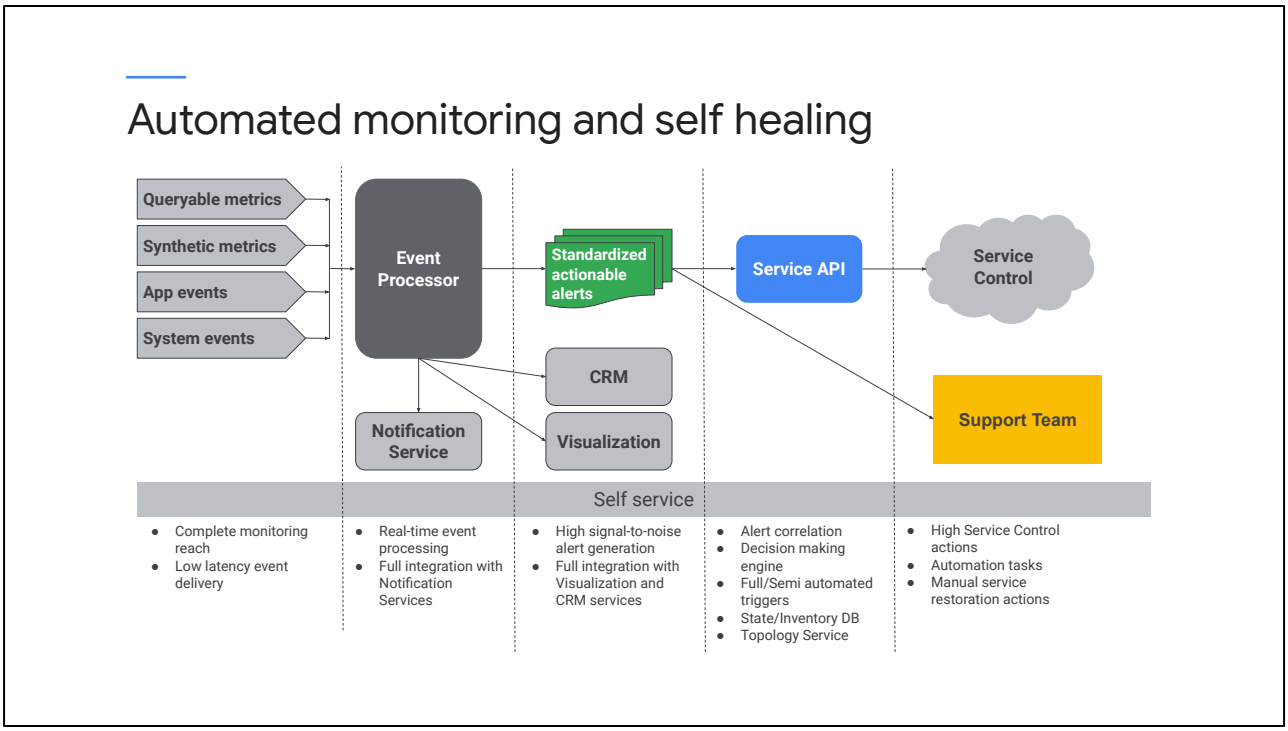
An alert that is generated by your monitoring system must have sufficient information that makes it actionable.

Details in the alert notification should include the metric that triggered the alert, service information, and other details that help the operator take corrective action.

A best practice is to create playbooks that document the problem, any triggers, and steps to resolve or escalate the issue. This helps to create a well-defined, repeatable process for operators to follow when similar issues arise in the future.

In general, sending a notification of problems is not enough. Defining actionable alerts and playbooks to resolve the problems with automation is required and should be part of your monitoring and troubleshooting strategy.

Automating already available and proven playbooks allows your most valuable resource—the engineers—to focus on more complex problems.



An efficient monitoring system should include a set of components that work together to monitor your platform.

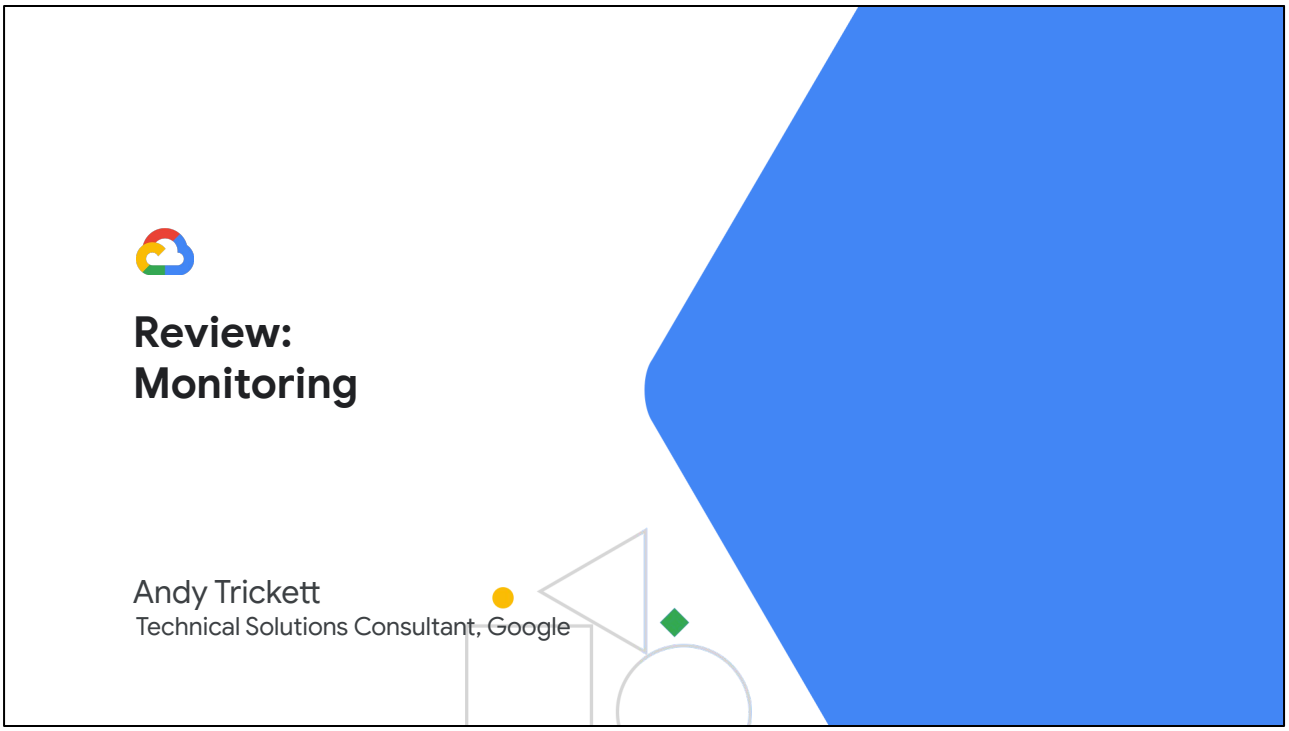
All metrics that help to maintain the service within the expected SLA are supplied to a real-time event processor that integrates with notification services and triggers standardized actionable alerts.

The event processor should also be able to asynchronously feed data to CRM and visualization services for reporting.

The alerts are then correlated and fed to a decision-making system that either triggers automated remediation steps or escalates to the support team.

All of these actions and any changes made to finally resolve the issue must be audited.

Audit trails can help create root cause analysis reports, which are used to make improvements in the system to mitigate future issues.



In this module, we discussed the practices used to monitor the components of the Apigee Edge API platform for private cloud.

We discussed the role of synthetic transactions as part of a monitoring strategy, and described ways to monitor the API proxies that are deployed to the platform.

We also discussed best practices to establish a scalable monitoring strategy in your organization.


Google Cloud