Monitoring and Telemetry

Prof. Ítalo Cunha

Telemetry is fundamental in cloud operations

- Monitor system behavior
- Evaluate performance
- Forecast demand
- Provision resources
- Troubleshoot failures
- Identify attacks
- Diagnose problems
- Perform audits
- Accurate billing

Metrics

- Quantitative data about a system
  - Link bandwidth, CPU utilization, memory usage
  - May be categorical: status (up/down), properties
- Collected periodically
  - Active: Run a task to collect a measurement (ping for RTT)
  - Passive: Read or receive data (CPU utilization or SNMP with link utilization)
- Metrics can be associated with
  - Physical resources: servers, switches, links
  - Virtual resources: VMs, containers, virtual disks, virtual networks
  - Abstractions: Deployment, load balancer, databases

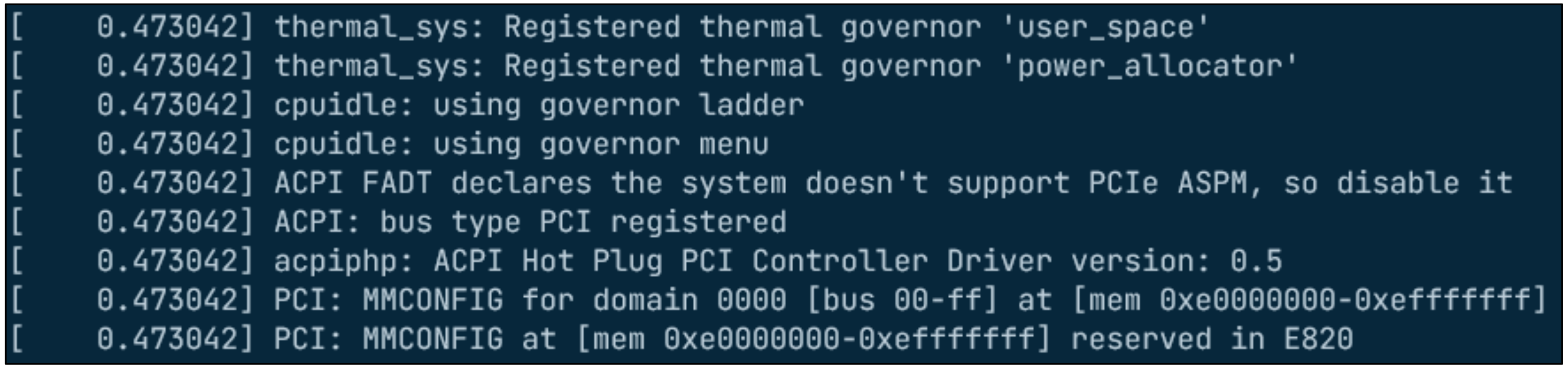
Logs

- Qualitative data about noteworthy events
  - Can be used proactively to identify issues and trigger alerts
  - More often used reactively to troubleshoot issues after detection
- Logging messages should have some standard to ease processing
  - Timestamp
  - Identification of the code generating the message

Logs

- Qualitative data about noteworthy events
  - Can be used proactively to identify issues and trigger alerts
  - More often used reactively to troubleshoot issues after detection
- Logging messages should have some standard to ease processing
  - Timestamp
  - Identification of the code generating the message

Linux dmesg

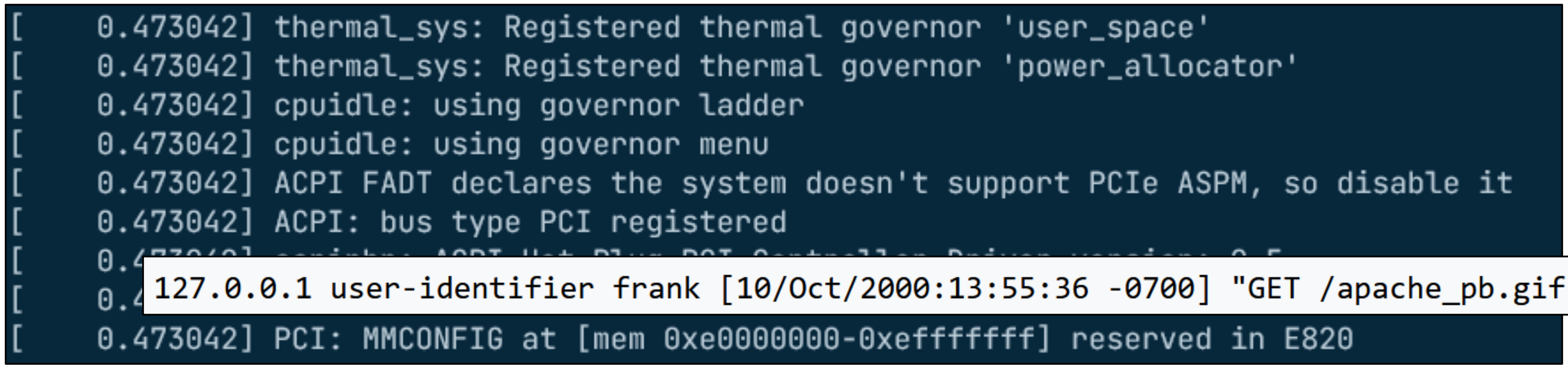


```
[    0.473042] thermal_sys: Registered thermal governor 'user_space'
[    0.473042] thermal_sys: Registered thermal governor 'power_allocator'
[    0.473042] cpuidle: using governor ladder
[    0.473042] cpuidle: using governor menu
[    0.473042] ACPI FADT declares the system doesn't support PCIe ASPM, so disable it
[    0.473042] ACPI: bus type PCI registered
[    0.473042] acpiphp: ACPI Hot Plug PCI Controller Driver version: 0.5
[    0.473042] PCI: MMCONFIG for domain 0000 [bus 00-ff] at [mem 0xe0000000-0xefffffff]
[    0.473042] PCI: MMCONFIG at [mem 0xe0000000-0xefffffff] reserved in E820
```

Logs

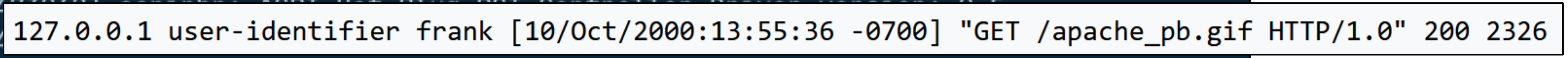
- Qualitative data about noteworthy events
  - Can be used proactively to identify issues and trigger alerts
  - More often used reactively to troubleshoot issues after detection
- Logging messages should have some standard to ease processing
  - Timestamp
  - Identification of the code generating the message

Linux dmesg



```
[    0.473042] thermal_sys: Registered thermal governor 'user_space'
[    0.473042] thermal_sys: Registered thermal governor 'power_allocator'
[    0.473042] cpuidle: using governor ladder
[    0.473042] cpuidle: using governor menu
[    0.473042] ACPI FADT declares the system doesn't support PCIe ASPM, so disable it
[    0.473042] ACPI: bus type PCI registered
[    0.4
[    0.4
[    0.473042] PCI: MMCONFIG at [mem 0xe0000000-0xefffffff] reserved in E820
```

Common log format



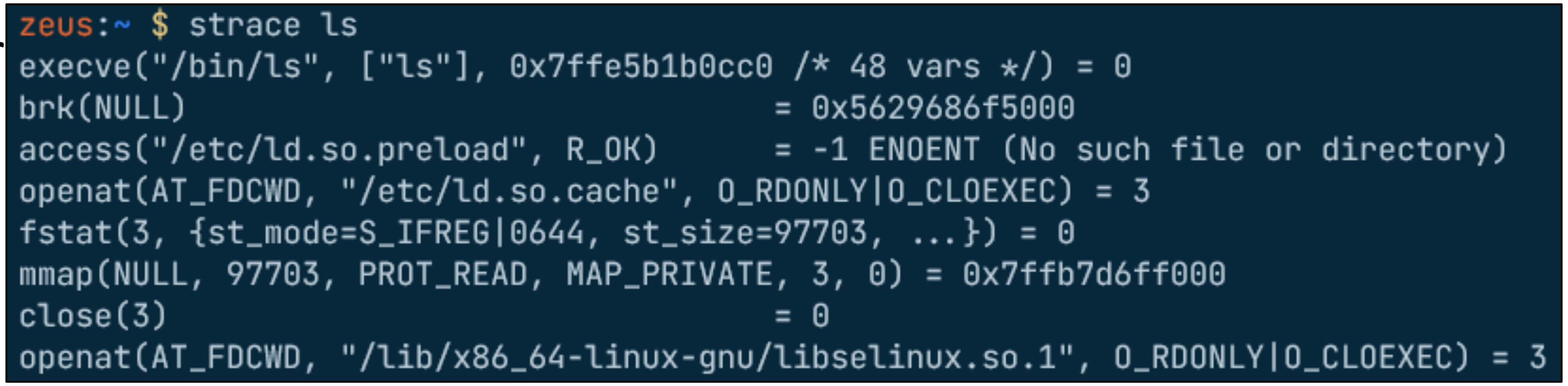
```
127.0.0.1 user-identifier frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

Traces

- Records of relationships during the execution of a task
  - Logs capture events
  - Traces capture the sequence and relationships between events or steps
  - Fine grained information
- Traces are collected across process boundaries
  - A distributed challenge
  - Interactions may not be known in advance
  - Synchronization across devices is important for troubleshooting time-dependent issues

Traces

- Records of relationships during the execution of a task
  - Logs capture events
  - Traces capture the sequence and relationships between events or steps
  - Fine grained information
- Traces are collected across process boundaries
  - A distributed challenge
  - Interactions may not be known in advance
  - Synchronization acr
    dependent issues



```
zeus:~ $ strace ls
execve("/bin/ls", ["ls"], 0x7ffe5b1b0cc0 /* 48 vars */) = 0
brk(NULL)                               = 0x5629686f5000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=97703, ...}) = 0
mmap(NULL, 97703, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ffb7d6ff000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
```

Telemetry use cases

Monitoring

- Ongoing measurements
- Proactive
- Needs to be lightweight
- Needs broad coverage
- Needs to be automated

Troubleshooting

- On-demand measurements
- Reactive
- May be expensive
- May be specific
- May require human oversight

Monitoring needs tooling to be useful

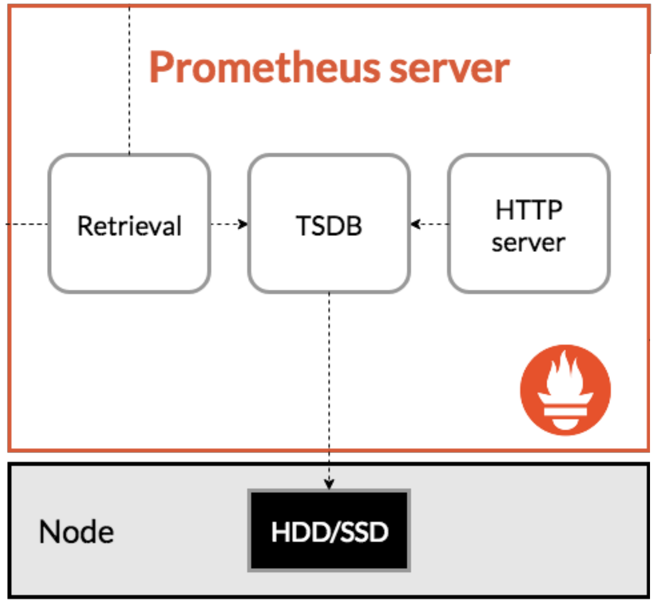
- Automated collection, processing, indexing, and storage
- Alerts to detect events or conditions
- Closed-loop controllers that can take actions depending on conditions
- Dashboards for human consumption

Monitoring needs tooling to be useful

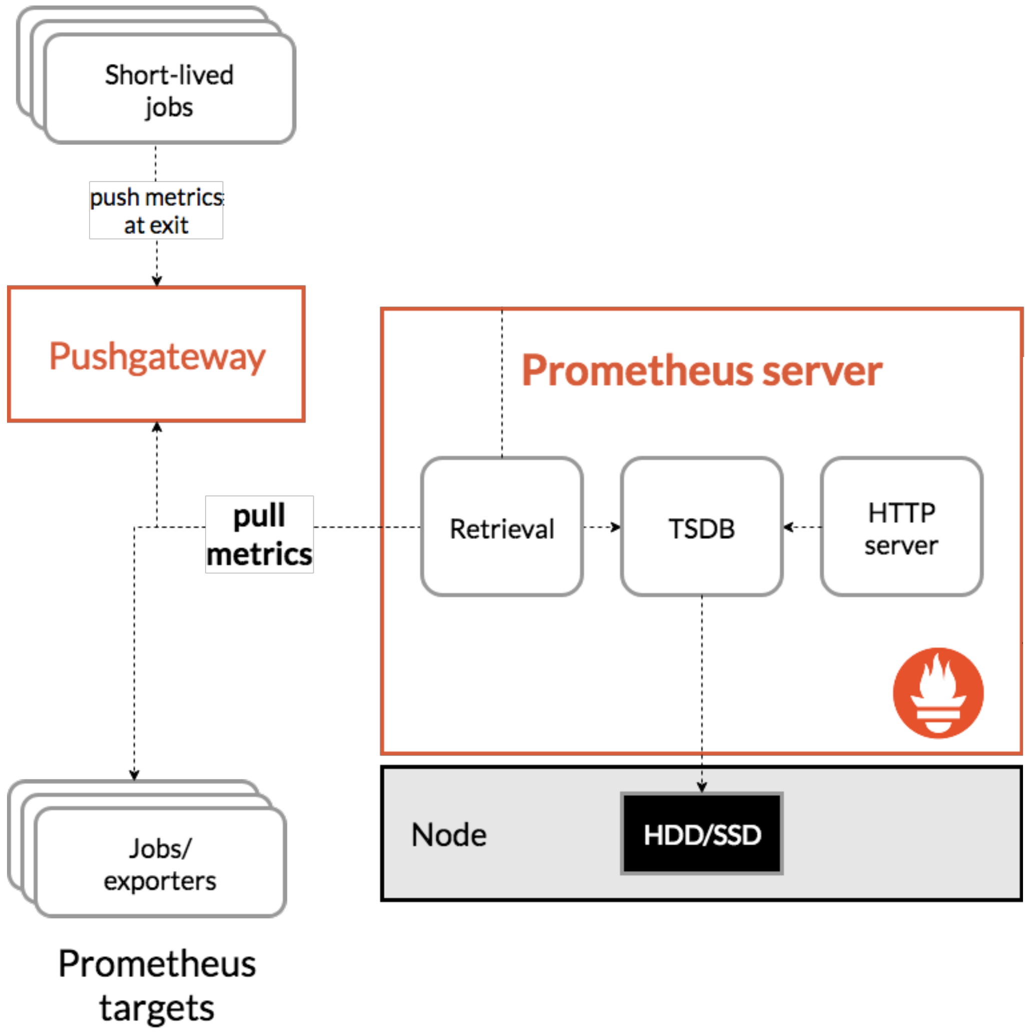
- Automated collection, processing, indexing, and storage
- Alerts to detect events or conditions
- Closed-loop controllers that can take actions depending on conditions
- Dashboards for human consumption

- Integration with machine learning for anomaly detection and inference

Prometheus

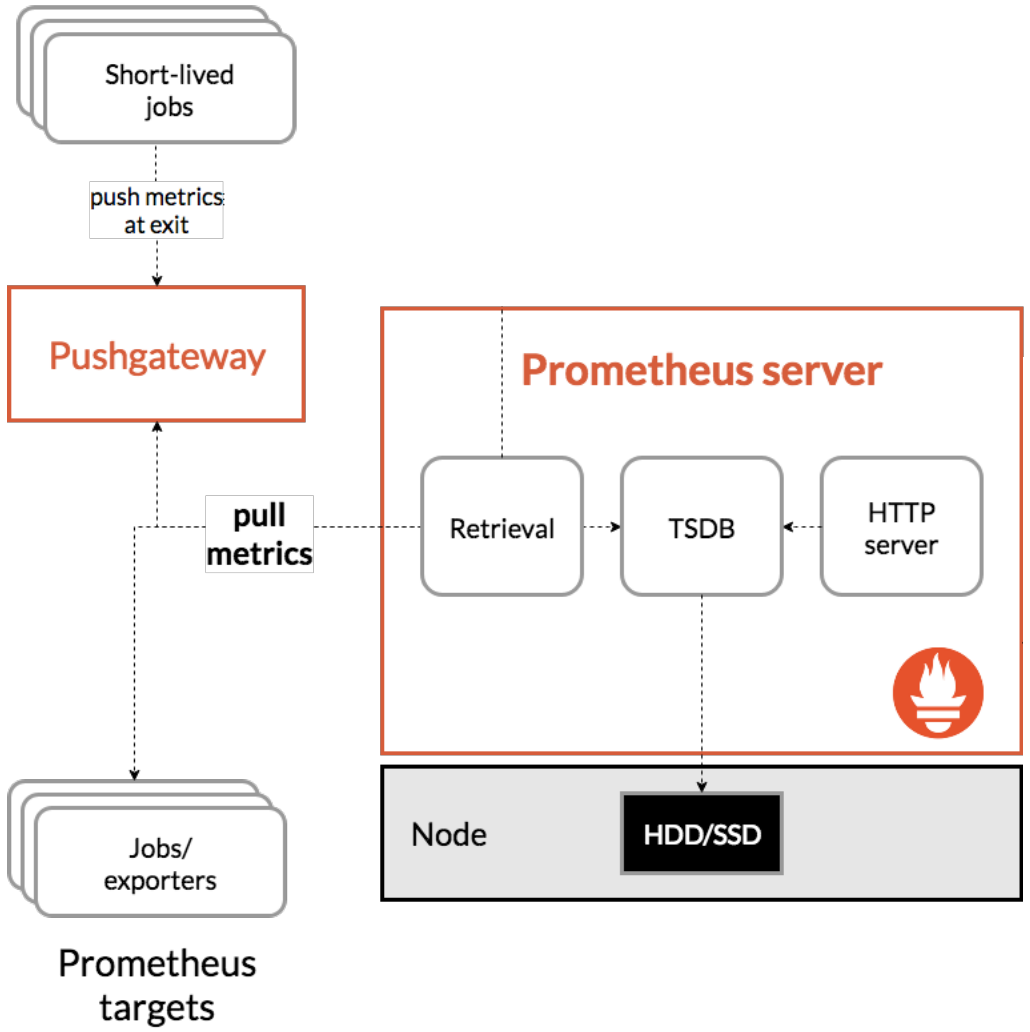


Prometheus



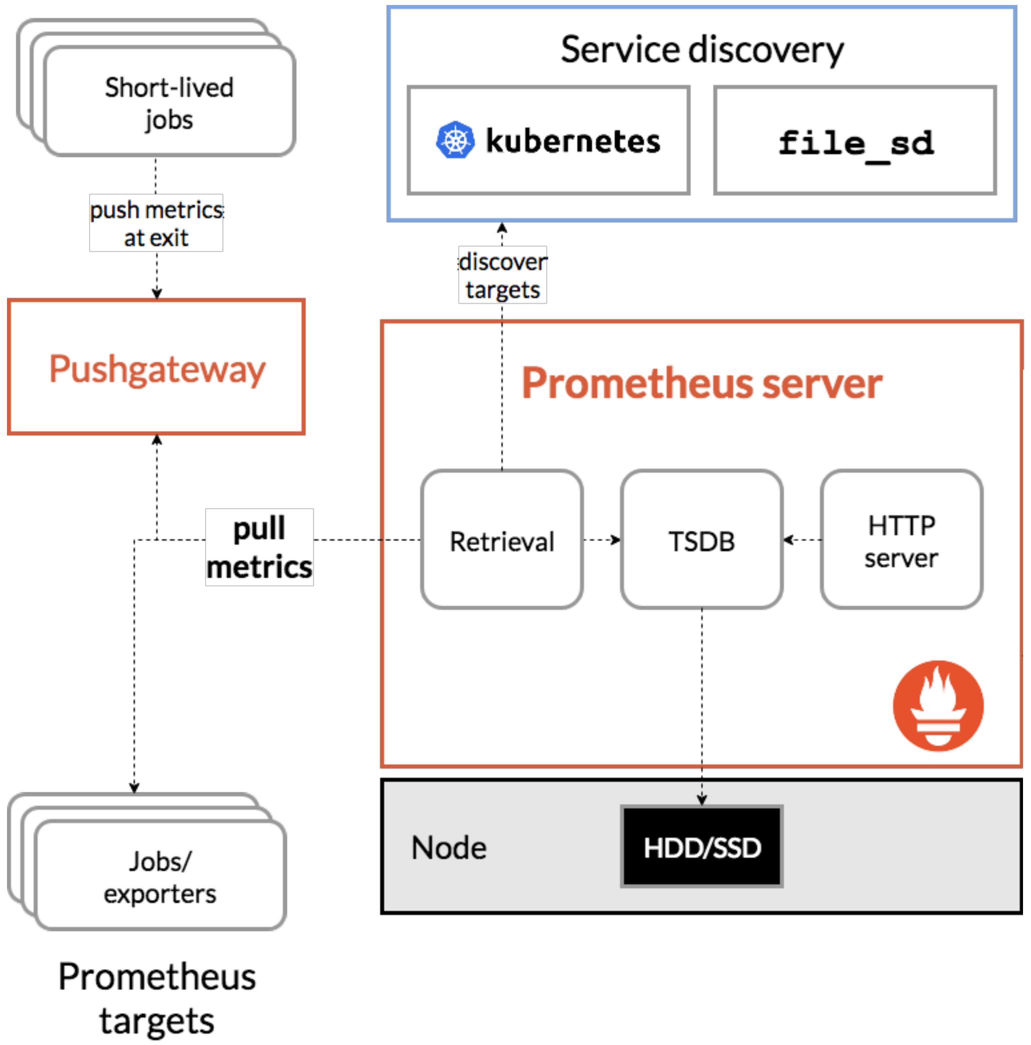
Short-lived jobs
push metrics at exit
Pushgateway
pull metrics
Prometheus server
Retrieval
TSDB
HTTP server
Node
HDD/SSD
Jobs/ exporters
Prometheus targets

Prometheus



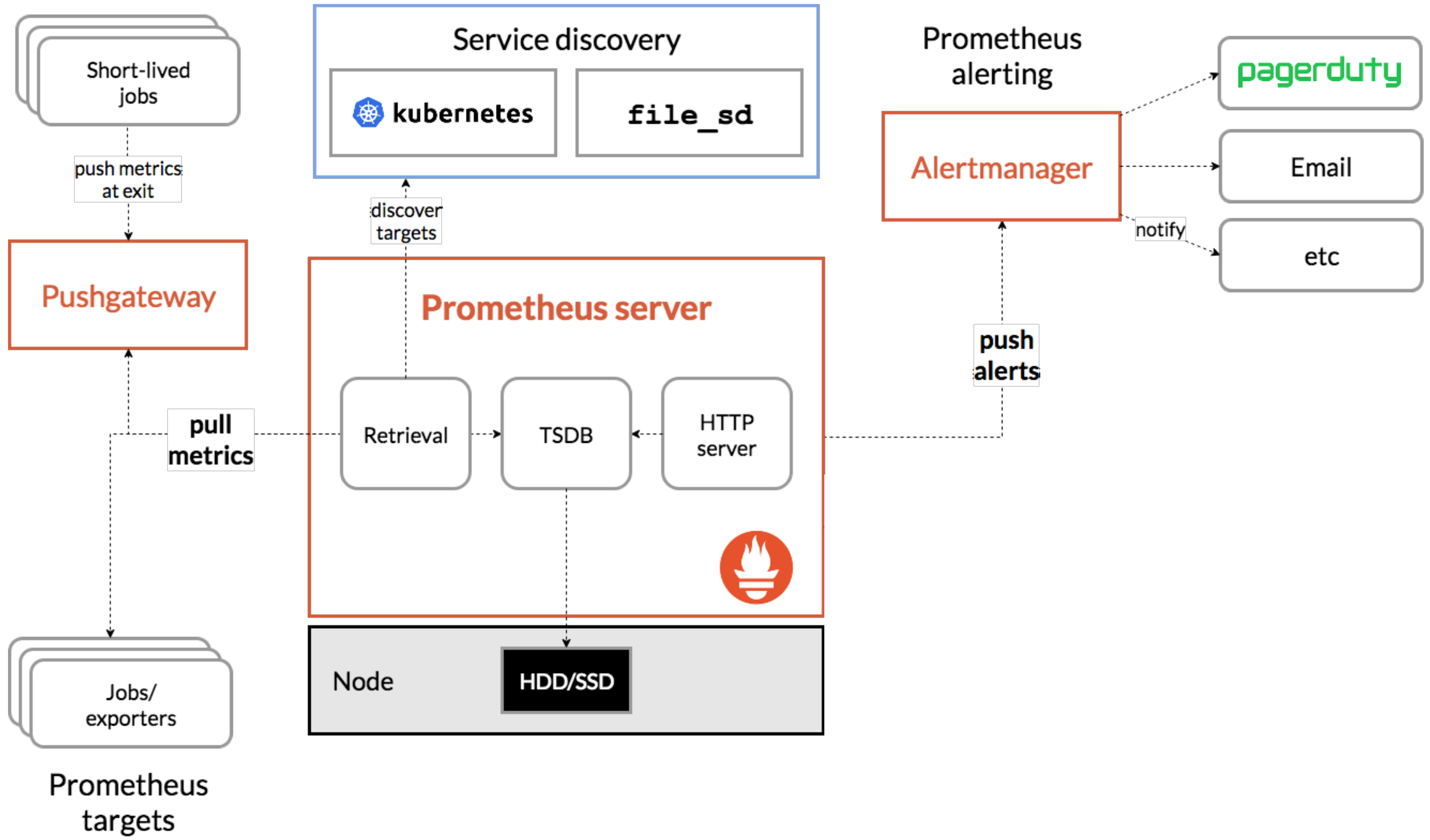
Client libraries

Prometheus



Client libraries

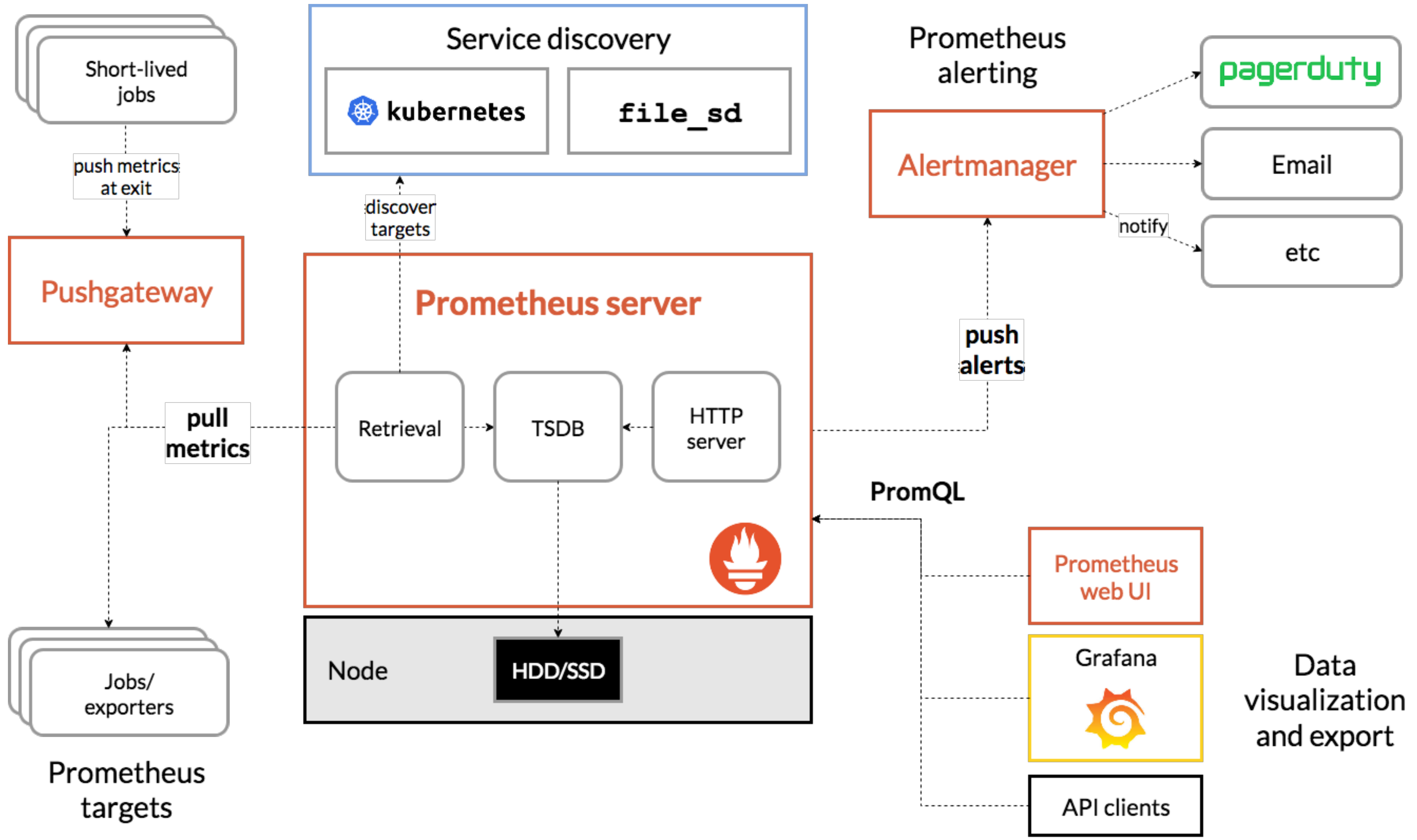
Prometheus



Plugin support

Client libraries

Prometheus


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

Handing distributed infrastructures

- Where to store monitoring and telemetry data?
  - Locally on each site vs centralized on a single site
- Trade-offs to consider
  - Possible use of bandwidth to transfer large amounts of data
  - Replication requires storage space
  - Accessing remote data may add delays

Handing distributed infrastructures

- Where to store monitoring and telemetry data?
  - Locally on each site vs centralized on a single site
- Trade-offs to consider
  - Possible use of bandwidth to transfer large amounts of data
  - Replication requires storage space
  - Accessing remote data may add delays

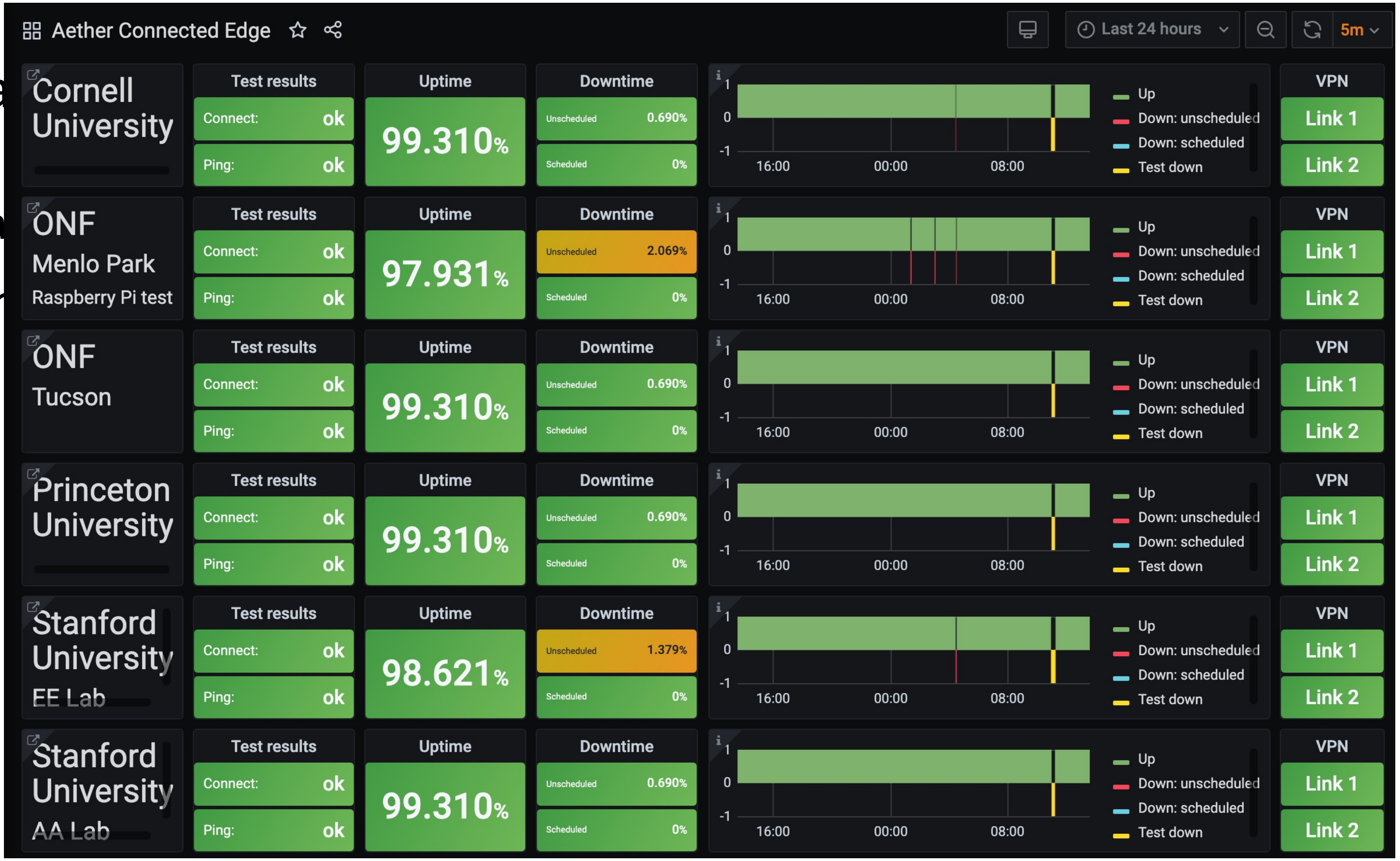
- Aether uses a mixes approach
  - Keeps raw data local but centralizes query results

Graphana

- Predefined visualizations for different types of data
  - Time series
  - Resource allocations and utilizations
- Integrations with telemetry frameworks and applications
- A Graphana deployment can pull information from multiple sources

Graphana

- Predefined visua
  - Time series
  - Resource alloca
- Integrations with
- A Graphana dep



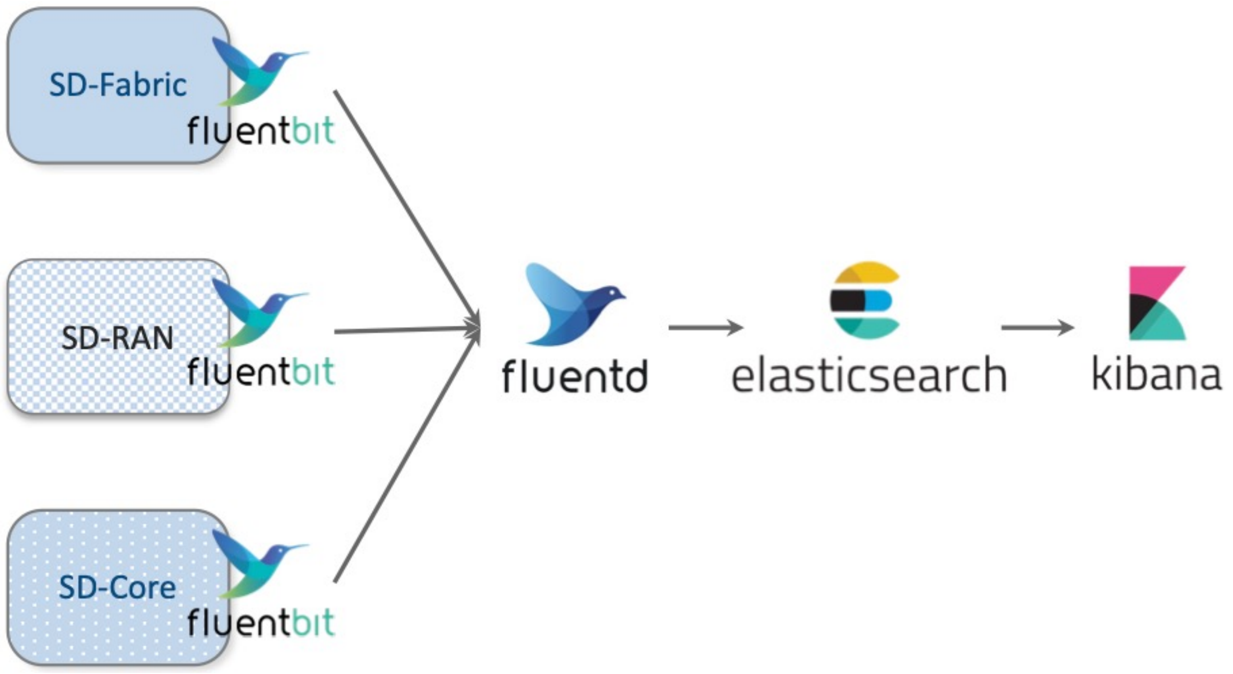
Aether Connected Edge

Last 24 hours 5m

Site	Test results		Uptime	Downtime		VPN	
Cornell University	Connect: ok	Ping: ok	99.310%	Unscheduled 0.690%	Scheduled 0%	Link 1	Link 2
ONF Menlo Park Raspberry Pi test	Connect: ok	Ping: ok	97.931%	Unscheduled 2.069%	Scheduled 0%	Link 1	Link 2
ONF Tucson	Connect: ok	Ping: ok	99.310%	Unscheduled 0.690%	Scheduled 0%	Link 1	Link 2
Princeton University	Connect: ok	Ping: ok	99.310%	Unscheduled 0.690%	Scheduled 0%	Link 1	Link 2
Stanford University EE Lab	Connect: ok	Ping: ok	98.621%	Unscheduled 1.379%	Scheduled 0%	Link 1	Link 2
Stanford University AA Lab	Connect: ok	Ping: ok	99.310%	Unscheduled 0.690%	Scheduled 0%	Link 1	Link 2

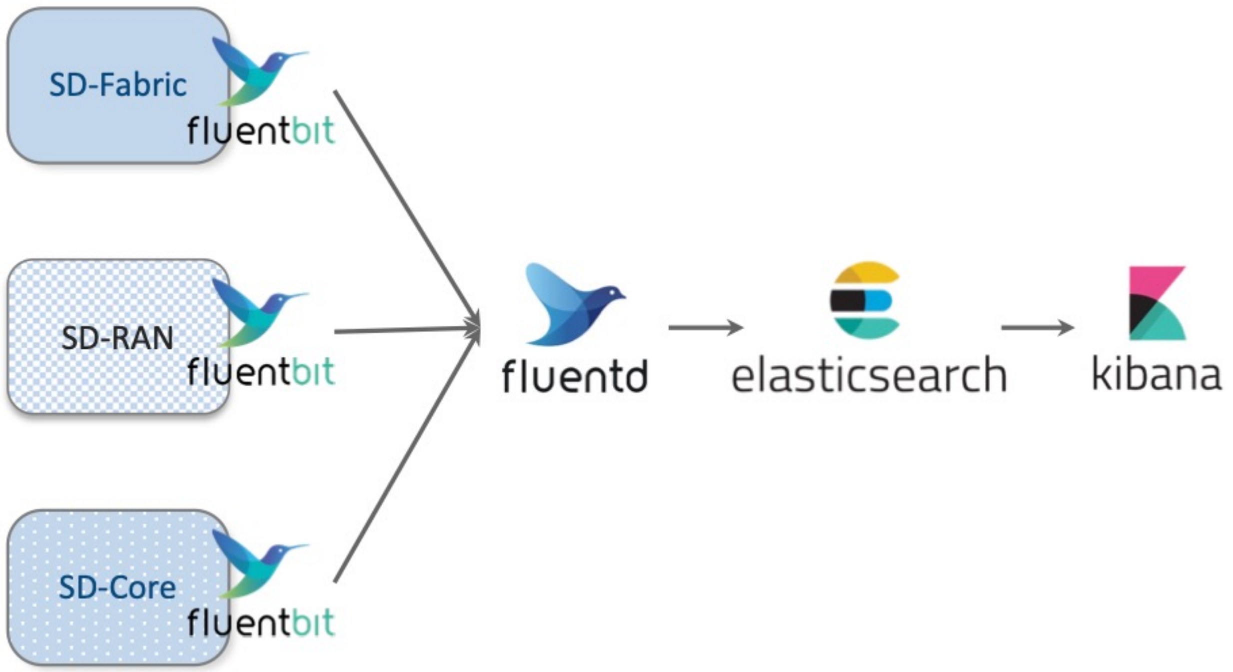
Logging for cloud-native applications

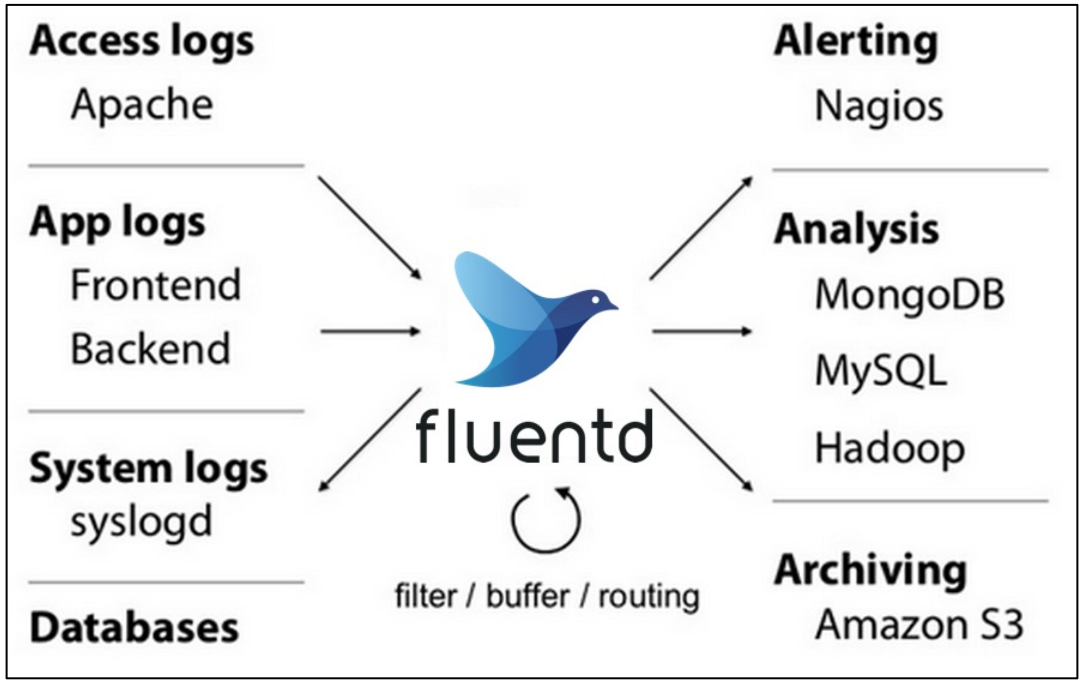
- Need to aggregate and correlate system and application logs



Logging for cloud-native applications

- Need to aggregate and correlate system and application logs





Logging for cloud-native applications

- Need to aggregate and correlate system and application logs
- Need to normalize logs

```
2020-08-18 05:35:54.842Z INFO [DistributedP4RuntimeTableMirror]
Synchronized TABLE_ENTRY mirror for device:leaf1: 0 removed, 2 updated, 4 added
```

⬇

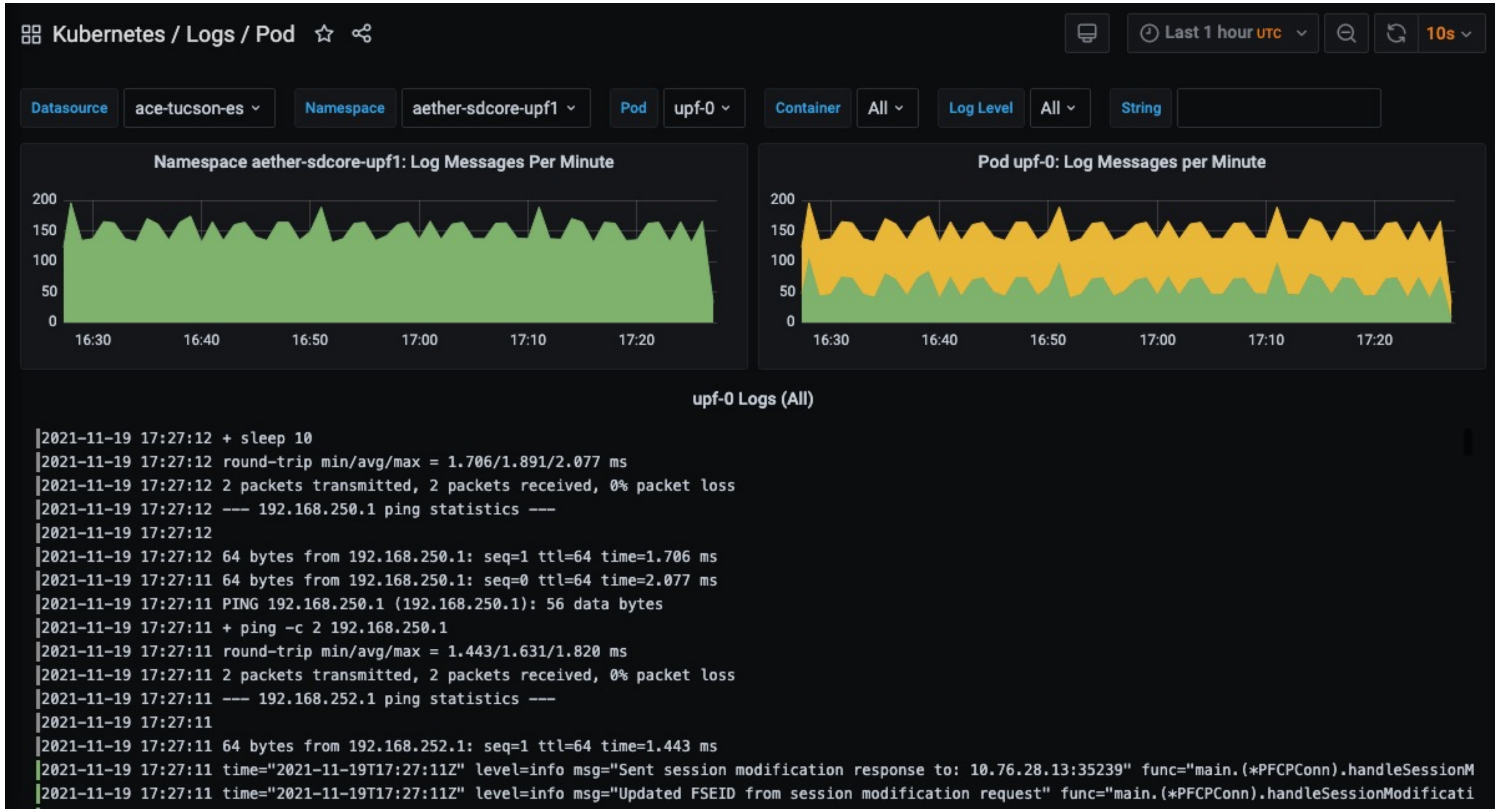
```
{
  "time": "2020-08-18 05:35:54.842Z",
  "logLevel": "INFO",  "component": "DistributedP4RuntimeTableMirror",
  "log": "Synchronized TABLE_ENTRY mirror for device:leaf1: 0 removed, 2 updated, 4 added"
}
```

Best practices

- Log shipping handled by the platform
  - Use if a logging library or stdout/stderr
- File logging should be disabled
  - Layered filesystems used in containers have poor write performance
  - Unnecessary if logging is handled by the platform
- Asynchronous logging is preferred
- Timestamps should be accurate
  - Stamp as close as possible to log event, keeps clocks synchronized
- Ability to change log levels at runtime
- Integrated dashboards

Integrated dashboards

- Combine monitoring, logging, and tracing in a single interface



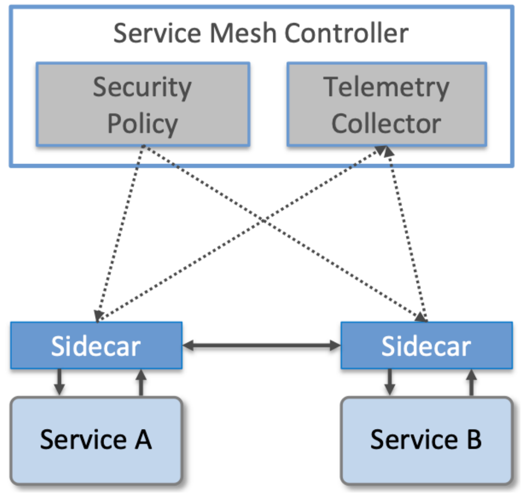
```
2021-11-19 17:27:12 + sleep 10
2021-11-19 17:27:12 round-trip min/avg/max = 1.706/1.891/2.077 ms
2021-11-19 17:27:12 2 packets transmitted, 2 packets received, 0% packet loss
2021-11-19 17:27:12 --- 192.168.250.1 ping statistics ---
2021-11-19 17:27:12
2021-11-19 17:27:12 64 bytes from 192.168.250.1: seq=1 ttl=64 time=1.706 ms
2021-11-19 17:27:11 64 bytes from 192.168.250.1: seq=0 ttl=64 time=2.077 ms
2021-11-19 17:27:11 PING 192.168.250.1 (192.168.250.1): 56 data bytes
2021-11-19 17:27:11 + ping -c 2 192.168.250.1
2021-11-19 17:27:11 round-trip min/avg/max = 1.443/1.631/1.820 ms
2021-11-19 17:27:11 2 packets transmitted, 2 packets received, 0% packet loss
2021-11-19 17:27:11 --- 192.168.252.1 ping statistics ---
2021-11-19 17:27:11
2021-11-19 17:27:11 64 bytes from 192.168.252.1: seq=1 ttl=64 time=1.443 ms
2021-11-19 17:27:11 time="2021-11-19T17:27:11Z" level=info msg="Sent session modification response to: 10.76.28.13:35239" func="main.(*PFCPConn).handleSessionM
2021-11-19 17:27:11 time="2021-11-19T17:27:11Z" level=info msg="Updated FSEID from session modification request" func="main.(*PFCPConn).handleSessionModificati
```

Tracing an application

- Need to collect fine-grained information from processes
- And keep track of requests across process boundaries
- Tracing frameworks integrate with existing code
  - Language runtimes (e.g., JVM, node.js)
  - Frameworks (e.g., Flask, Spring, ASP.NET)

Observability

- New buzzword for monitoring and telemetry
- Denotes a system that follows the best practices we have discussed
  - Many recent innovations in this area, systems need to integrate
  - Software frameworks for host-based monitoring
  - Service meshes
  - Software-defined networks with programmable, on-path monitoring


Service Mesh Controller
Security Policy
Telemetry Collector
Sidecar
Sidecar
Service A
Service B