Logging and Monitoring in CORD

Zack Williams - zdw@opennetworking.org

Goals

Answers the operational question: *What is the system doing?*

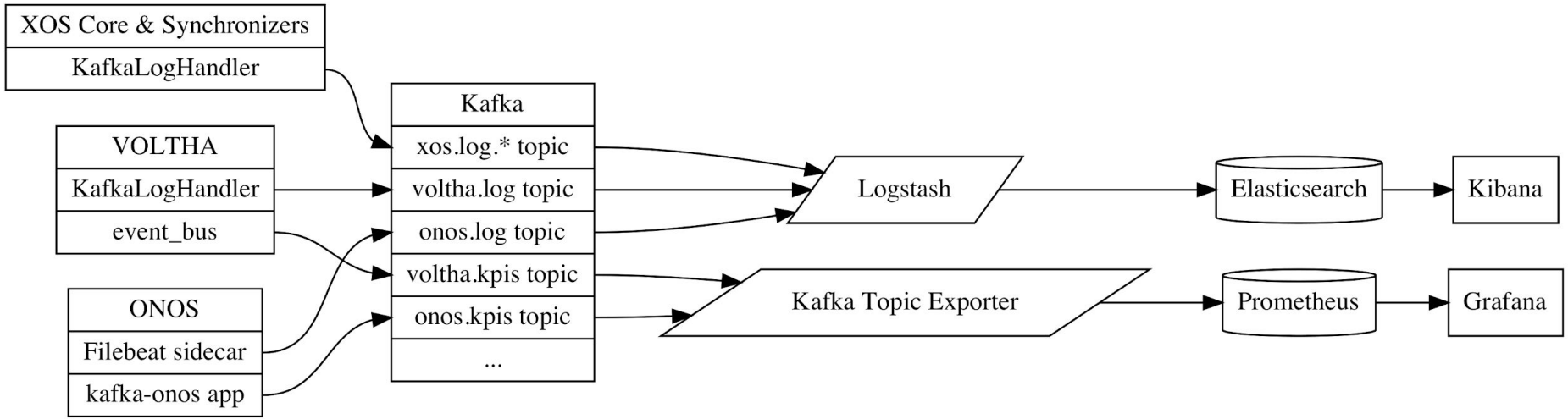
- Collects statistics, actionable items, and other monitoring data
- Provides a historical view of the system to aid in diagnostics and troubleshooting
- Lower the barrier to entry for extending the system, allowing new and novel uses of data

FCAPS

Logging / Monitoring

- **Fault Management**
- Config - handled in XOS and via other APIs
- **Accounting**
- **Performance**
- **Security**

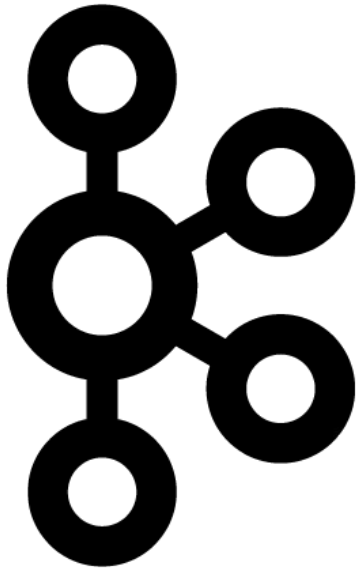
The Big Picture



Kafka

Message bus, used for two purposes:

- Pass actionable messages between components
- Collect messages (logging + monitoring)



Why Kafka?

- Already used in VOLTHA and XOS
- Performs and scales well
- API bindings across multiple languages

Kafka Event Concepts

Topics: Namespaces that contain sets of events

Keys: Summarizable (last item retained on vacuum) per-topic categories

Producers: Adds messages to a topic

Consumer: Reads messages from a topic

Offset: The location last read by a consumer in a specific topic, allows multiple consumers of a topic to split load

Kafka use as an event bus in CORD

Events happen in XOS, ONOS, and VOLTHA

New ONU devices are turned on in VOLTHA causes ONOS to creates an event on the `onu.events` topic.

K8s pod starts, XOS Kubernetes synchronizer creates an event on the `xos.kubernetes.pod-details` topic, which the XOS ONOS synchronizer uses to know if it needs to reload configuration into ONOS if that pod is restarted.

Logging

Provides answers to the 5 W's:

Who: The component that created the log message

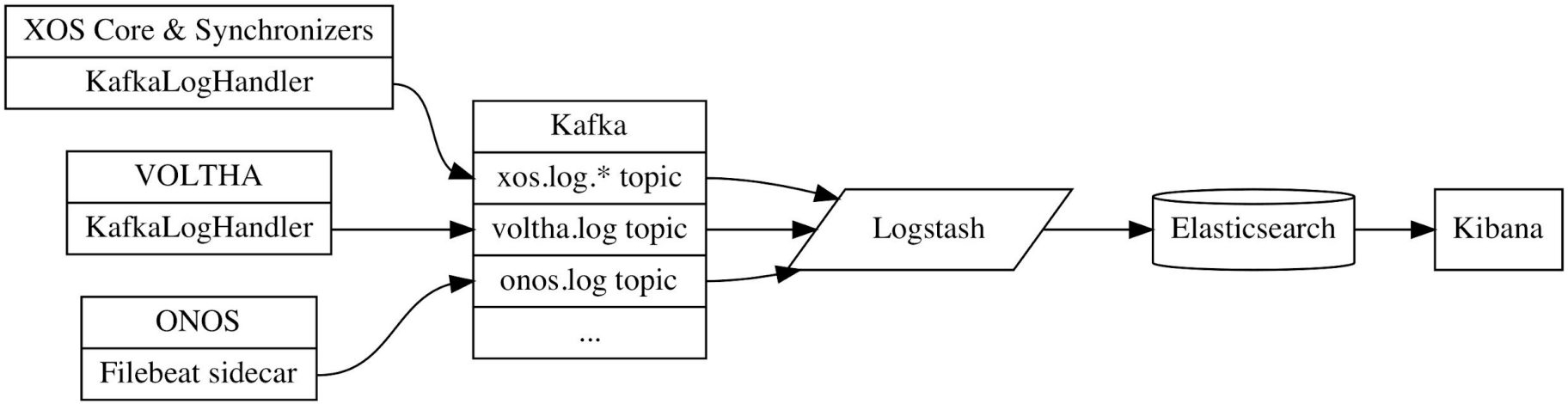
What: Content of the message

Where: Gives the context in the codebase

When: A precise time the log message was created

Why: Determined from message + context

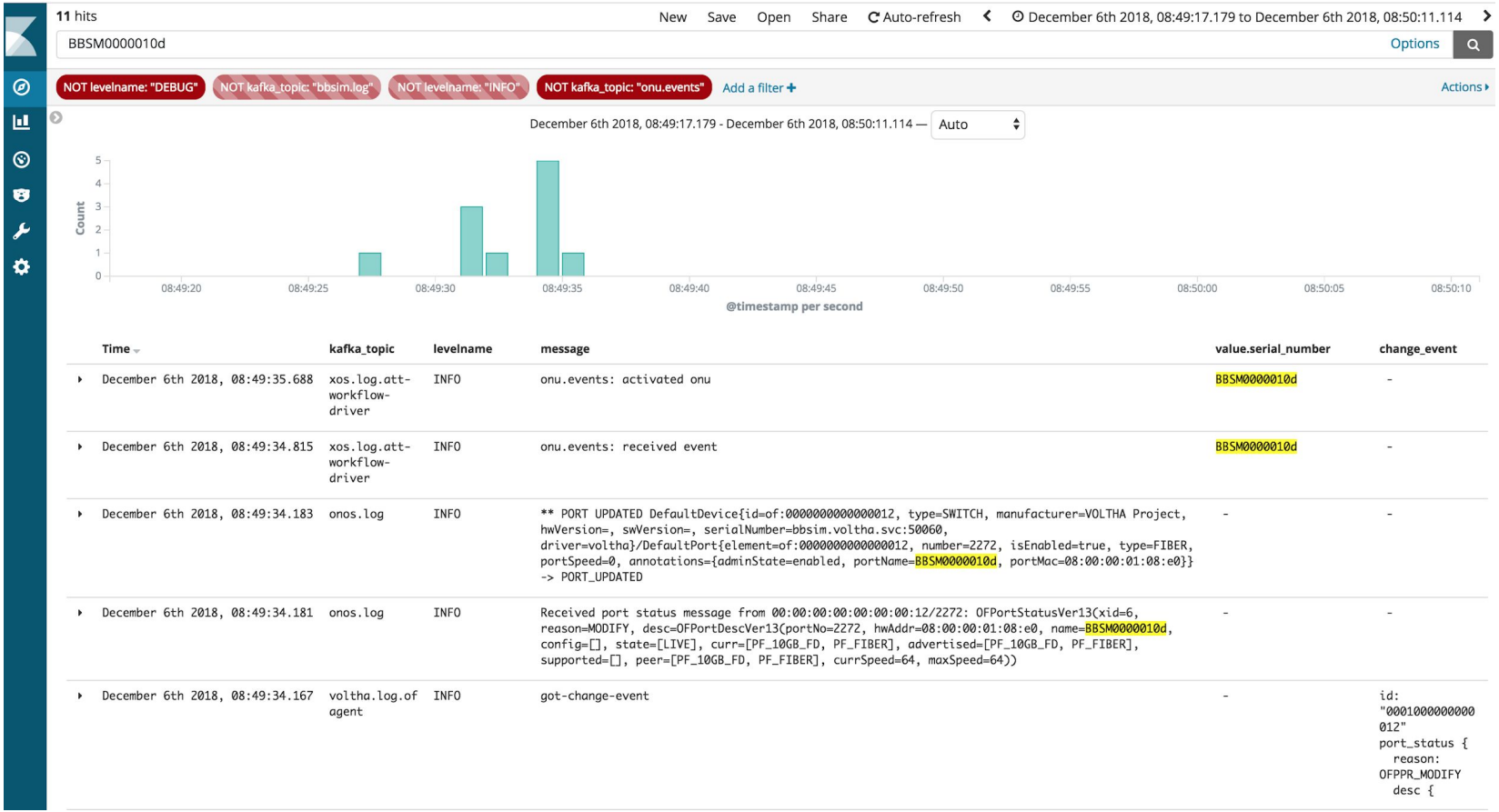
Logging Pipeline



In CORD/SEBA, logs are put on `*.log.*` topic in Kafka

Logstash consumes from Kafka and adds to Elasticsearch, displayed via Kibana web UI

Logging Visualization



Time	kafka_topic	levelname	message	value.serial_number	change_event
December 6th 2018, 08:49:35.688	xos.log.att-workflow-driver	INFO	onu.events: activated onu	BBSM0000010d	-
December 6th 2018, 08:49:34.815	xos.log.att-workflow-driver	INFO	onu.events: received event	BBSM0000010d	-
December 6th 2018, 08:49:34.183	onos.log	INFO	** PORT UPDATED DefaultDevice{id=of:0000000000000012, type=SWITCH, manufacturer=VOLTHA Project, hwVersion=, swVersion=, serialNumber=bbsim.voltha.svc:50060, driver=voltha}/DefaultPort{element=of:0000000000000012, number=2272, isEnabled=true, type=FIBER, portSpeed=0, annotations={adminState=enabled, portName=BBSM0000010d, portMac=08:00:00:01:08:e0}} -> PORT_UPDATED	-	-
December 6th 2018, 08:49:34.181	onos.log	INFO	Received port status message from 00:00:00:00:00:00:00:12/2272: OFPortStatusVer13(xid=6, reason=MODIFY, desc=OFPortDescVer13(portNo=2272, hwAddr=08:00:00:01:08:e0, name=BBSM0000010d, config=[], state=[LIVE], curr=[PF_10GB_FD, PF_FIBER], advertised=[PF_10GB_FD, PF_FIBER], supported=[], peer=[PF_10GB_FD, PF_FIBER], currSpeed=64, maxSpeed=64))	-	-
December 6th 2018, 08:49:34.167	voltha.log.of agent	INFO	got-change-event	-	id: "0001000000000012" port_status { reason: OFPPR_MODIFY desc {

Structured Logging

Logging but with "more data" added on, usually key/value pairs

Framework automatically adds a Who, Where, and When

- Timestamps, instance name, filename and line number of logging call, etc.

What/Why are enhanced by extra fields, and can help with log correlation

Structured Logging Flow

Python Code:

```
self.logger.debug("MODEL_POLICY: updating subscriber",onu_device=subscriber.onu_device,
authentication_state=si.authentication_state, subscriber_status=subscriber.status)
```

Kafka Message:


```
{"relativeCreated":1014018.0199146271,"process":1,"@timestamp":"2018-12-06T15:53:58.003470Z","module":"model_
policy_att_workflow_driver_serviceinstance","funcName":"update_subscriber","threadId":140183227852544,"messag
e":"MODEL_POLICY: updating
subscriber","filename":"model_policy_att_workflow_driver_serviceinstance.py","levelno":10,"processName":"Main
Process","lineno":128,"subscriber_status":"enabled","onu_device":"BBSM0000010a","args":[],"authentication_sta
te":"APPROVED","exc_text":null,"name":"model_policy_att_workflow_driver_whitelistentry","threadName":"policy_
engine","msecs":3.4699440002441406,"pathname":"/opt/xos/synchronizers/att-workflow-driver/model_policies/mode
l_policy_att_workflow_driver_serviceinstance.py","exc_info":null,"levelname":"DEBUG"}
```


Kibana:

Time	kafka_topic	message	levelname	onu_device	authentication_state	subscriber_status
December 6th 2018, 08:53:58.003	xos.log.att-workflow-driver	MODEL_POLICY: updating subscriber	DEBUG	BBSM0000010a	APPROVED	enabled

Guidelines for writing log messages

Think like a reader when you write the message

- Be precise and descriptive
- "Don't repeat yourself" also applies to messages

Don't capture security or human generated information

- Incorrect password attempt, security certificates
- Any user input that could be identifiable
- Anonymous metadata/statistics is OK

Logging Levels

Too many log messages becomes an overwhelming, "needle in a haystack" problem - log levels help with this.

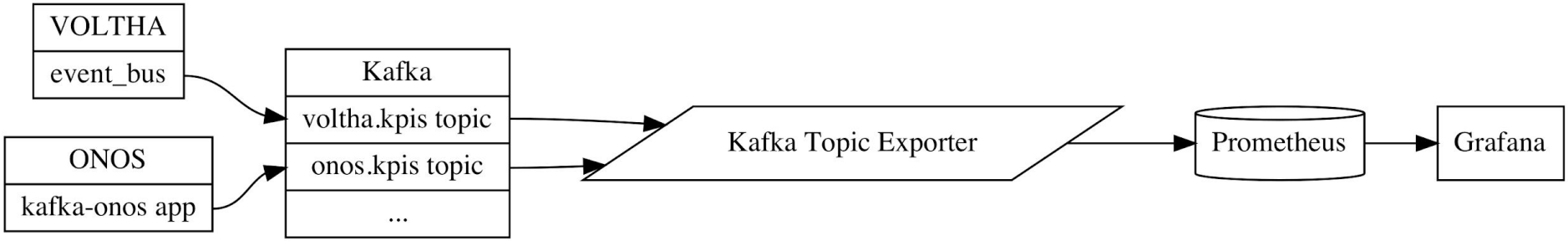
`ERROR` or `FATAL` == user intervention required

- Exception logging is at `ERROR` level, so catch these if Exceptions on transient issues, and log at `WARNING`

Use `INFO` or `DEBUG` (or `TRACE`, if available) for other messages

In production, usually only `INFO` level is captured.

Monitoring Pipeline



Kafka Topic Exporter consumes from Kafka, adds to Prometheus time-series database, displayed with Grafana

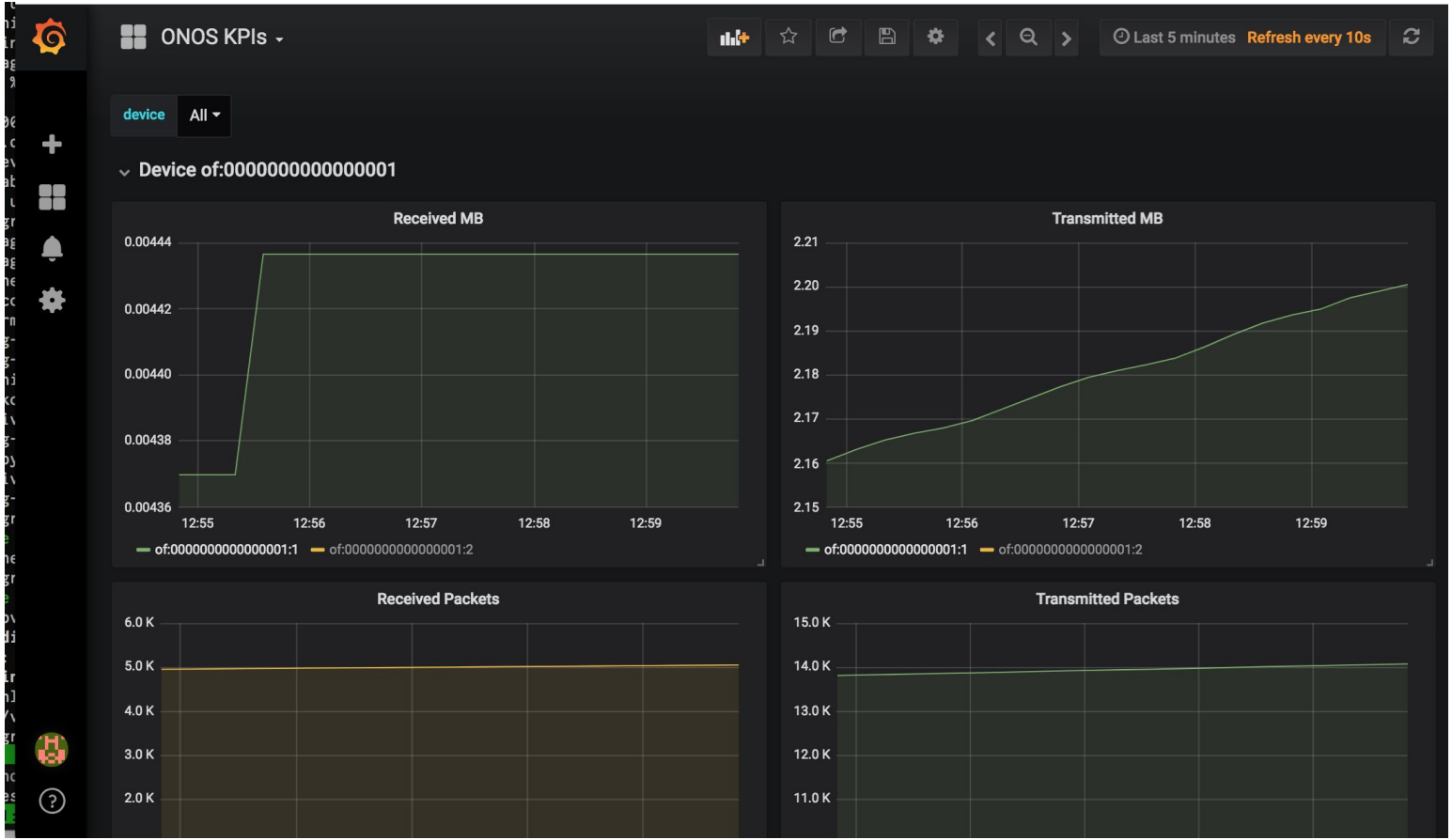
Monitoring Format

Designed with prometheus in mind - example from `onos.kpis`:


```
{"timestamp":"2018-12-06T20:07:28.733Z","ports":[{"portId":"1","pktRx":61,"pktTx":14503,"bytesRx":4722,"bytesTx":2380335,"pktRxDrp":6,"pktTxDrp":0},{"portId":"2","pktRx":5198,"pktTx":10374,"bytesRx":519026,"bytesTx":902538,"pktRxDrp":0,"pktTxDrp":0}],"deviceId":"of:0000000000000001"}
```

```
{"timestamp":"2018-12-06T20:07:33.732Z","ports":[{"portId":"1","pktRx":61,"pktTx":14509,"bytesRx":4722,"bytesTx":2381397,"pktRxDrp":6,"pktTxDrp":0},{"portId":"2","pktRx":5200,"pktTx":10378,"bytesRx":519226,"bytesTx":902886,"pktRxDrp":0,"pktTxDrp":0}],"deviceId":"of:0000000000000001"}
```

Monitoring Visualization



Extending Logging and Monitoring

If latency or RTT is an issue, interact with Kafka bus **within the pod**

If data is needed **outside the pod**, consume from Kafka, format as needed, and forward externally

- `ves-agent` is one example, specific to AT&T's and ONAP

If new/different information is needed, **contribute publishers** to Kafka, then consume as above.

Future Work

Normalize fields in structured log messages across components

Better namespacing of topics by their producing component

- `onu.events` -> `onos.events.onu`

Additional publishers/consumers of Kafka bus

Questions?