OCCI monitoring extension

and its proof of concept

Augusto Ciuffoletti

Dept. of Computer Science - Univ. of Pisa

September 5, 2014

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI framework

- There is an interface between the user of a cloud service and the cloud service itself
- Data entities that describe the service traverse this interface during its provisioning
- The protocol used during this conversation follows the REST paradigm:
  - the user plays the role of the client
  - the conversation follows the HTTP protocol
  - responses are cacheable, as far as possible
- OCCI proposes a minimalistic conceptual framework (or ontology) for the entities used to describe the service

The OCCI core concepts

- Anything is an **entity**, and is identified with an URI
- A relationship between entities is an entity
  - we distinguish **resource** entities and **link** entities (relationship)
- There are many **kinds** of entities, with distinguishing **attributes**
- An entity of a certain kind can be integrated with **mixins** that carry more attributes or bind existing ones

The OCCI core concepts

- Anything is an **entity**, and is identified with an URI
- A relationship between entities is an entity
  - we distinguish **resource** entities and **link** entities (relationship)
- There are many **kinds** of entities, with distinguishing **attributes**
- An entity of a certain kind can be integrated with **mixins** that carry more attributes or bind existing ones

The OCCI core concepts

- Anything is an **entity**, and is identified with an URI
- A relationship between entities is an entity
  - we distinguish **resource** entities and **link** entities (relationship)
- There are many **kinds** of entities, with distinguishing **attributes**
- An entity of a certain kind can be integrated with **mixins** that carry more attributes or bind existing ones

The OCCI core concepts

- Anything is an **entity**, and is identified with an URI
- A relationship between entities is an entity
  - we distinguish **resource** entities and **link** entities (relationship)
- There are many **kinds** of entities, with distinguishing **attributes**
- An entity of a certain kind can be integrated with **mixins** that carry more attributes or bind existing ones

The OCCI core concepts

- Anything is an **entity**, and is identified with an URI
- A relationship between entities is an entity
  - we distinguish **resource** entities and **link** entities (relationship)
- There are many **kinds** of entities, with distinguishing **attributes**
- An entity of a certain kind can be integrated with **mixins** that carry more attributes or bind existing ones

OCCI IaaS interface

- The first use case for OCCI, adopted as an open standard
  - **Resources** are processors, storage, networks.
  - **Links** are network interfaces, and processor/storage associations
  - **Mixins** add OpSys attributes to a processor, a filesystem to a storage, etc.
- However OCCI is designed to smoothly adapt to diverse use cases
- Here we want to propose its application to **monitoring the performance of cloud resources**

OCCI IaaS interface

- The first use case for OCCI, adopted as an open standard
  - **Resources** are processors, storage, networks.
  - **Links** are network interfaces, and processor/storage associations
  - **Mixins** add OpSys attributes to a processor, a filesystem to a storage, etc.
- However OCCI is designed to smoothly adapt to diverse use cases
- Here we want to propose its application to **monitoring the performance of cloud resources**

OCCI IaaS interface

- The first use case for OCCI, adopted as an open standard
  - **Resources** are processors, storage, networks.
  - **Links** are network interfaces, and processor/storage associations
  - **Mixins** add OpSys attributes to a processor, a filesystem to a storage, etc.
- However OCCI is designed to smoothly adapt to diverse use cases
- Here we want to propose its application to **monitoring the performance of cloud resources**

OCCI IaaS interface

- The first use case for OCCI, adopted as an open standard
  - **Resources** are processors, storage, networks.
  - **Links** are network interfaces, and processor/storage associations
  - **Mixins** add OpSys attributes to a processor, a filesystem to a storage, etc.
- However OCCI is designed to smoothly adapt to diverse use cases
- Here we want to propose its application to **monitoring the performance of cloud resources**

OCCI IaaS interface

- The first use case for OCCI, adopted as an open standard
  - **Resources** are processors, storage, networks.
  - **Links** are network interfaces, and processor/storage associations
  - **Mixins** add OpSys attributes to a processor, a filesystem to a storage, etc.
- However OCCI is designed to smoothly adapt to diverse use cases
- Here we want to propose its application to **monitoring the performance of cloud resources**

OCCI IaaS interface

- The first use case for OCCI, adopted as an open standard
  - **Resources** are processors, storage, networks.
  - **Links** are network interfaces, and processor/storage associations
  - **Mixins** add OpSys attributes to a processor, a filesystem to a storage, etc.
- However OCCI is designed to smoothly adapt to diverse use cases
- Here we want to propose its application to **monitoring the performance of cloud resources**

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
- Standardization matters!
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to certify the quality of the service
  - major cloud providers provides offer resource monitoring as a part of the service
- Standardization matters!
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers provides offer resource monitoring as a part of the service
- Standardization matters!
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
  - Consider the case of a composite service (many providers)
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
  - Consider the case of a composite service (many providers)
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
  - Consider the case of a composite service (many providers)
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
  - Consider the case of a composite service (many providers)
- We propose a simple ontology for resource monitoring that is aligned with OCCI

Motivation

- The monitoring of resource performance is a key issue to implement:
  - Service Level Agreement, a defined target to obtain user confidence
  - fault-tolerance targets defined by the user
- The user wants to customize resource monitoring
  - the user may be in his turn a service provider
  - the user may simply want to verify the quality of the service
  - major cloud providers providers offer resource monitoring as a part of the service
- Standardization matters!
  - Consider the case of a composite service (many providers)
- We propose a simple ontology for resource monitoring that is aligned with OCCI

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
- The first step entails the collaboration among entities
- The resource kind is named **Sensor**, and the link kind **Collector**
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
- The first step entails the collaboration among entities
- The resource kind is named **Sensor**, and the link kind **Collector**
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new Resource kind
- The first step entails the collaboration among entities
- The resource kind is named **Sensor**, and the link kind **Collector**
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
- The resource kind is named **Sensor**, and the link kind **Collector**
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new link kind
- The resource kind is named **Sensor**, and the link kind **Collector**
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
  - and this is bare minimum for a standalone monitoring framework
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
  - and this is bare minimum for a stand-alone monitoring framework
- The OCCI monitoring framework complements any OCCI framework

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
  - and this is bare minimum for a stand-alone **monitoring framework**
- The OCCI monitoring framework complements any OCCI framework
  - It handles any type of Resource

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
  - and this is bare minimum for a stand-alone **monitoring framework**
- The OCCI monitoring framework complements any OCCI framework
  - it handles any type of **Resource**

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
  - and this is bare minimum for a stand-alone **monitoring framework**
- The OCCI monitoring framework complements any OCCI framework
  - it handles any type of **Resource**.

A monitoring framework

- Monitoring is made of three basic activities:
  - extract performance parameters from the target **Resource**
  - aggregate them and compute the metric of interest
  - deliver the measurement to the relevant party
- The last two steps consist of aggregation and rendering of data
  - this makes a candidate for a new **Resource** kind
- The first step entails the collaboration among entities
  - this makes a candidate for a new **Link** kind
- The resource kind is named **Sensor**, and the link kind **Collector**
  - and this is bare minimum for a stand-alone **monitoring framework**
- The OCCI monitoring framework complements any OCCI framework
  - it handles any type of **Resource**.

A **Sensor**

- It is a distiguished activity that needs the provision of cloud resources
  - Tightly integrated in cloud infrastructure
  - Under control of the provider
  - Tuned using user requests
- The provider defines the available aggregation and publishing capabilities
- The user instantiates the Sensor as a composition of such capabilities

A **Sensor**

- It is a distiguished activity that needs the provision of cloud resources
  - Tightly integrated in cloud infrastructure
  - Under control of the provider
  - Tuned using user requests
- The provider defines the available aggregation and publishing capabilities
- The user instantiates the Sensor as a composition of such capabilities

A **Sensor**

- It is a distiguished activity that needs the provision of cloud resources
  - Tightly integrated in cloud infrastructure
  - Under control of the provider
  - Tuned using user requests
- The provider defines the available aggregation and publishing capabilities
- The user instantiates the Sensor as a composition of such capabilities

A **Sensor**

- It is a distiguished activity that needs the provision of cloud resources
  - Tightly integrated in cloud infrastructure
  - Under control of the provider
  - Tuned using user requests
- The provider defines the available aggregation and publishing capabilities
- The user instantiates the Sensor as a composition of such capabilities

A **Sensor**

- It is a distiguished activity that needs the provision of cloud resources
  - Tightly integrated in cloud infrastructure
  - Under control of the provider
  - Tuned using user requests
- The provider defines the available aggregation and publishing capabilities
- The user instantiates the Sensor as a composition of such capabilities

A **Sensor**

- It is a distiguished activity that needs the provision of cloud resources
  - Tightly integrated in cloud infrastructure
  - Under control of the provider
  - Tuned using user requests
- The provider defines the available aggregation and publishing capabilities
- The user instantiates the Sensor as a composition of such capabilities

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as mixins for the sensor
  - How data are aggregated (flow pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins
  - however the hook to connect a Sensor to a Resource must be defined

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins
  - however the hooks to connect a Sensor to a Collector must be defined

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins
  - however the hooks to connect a Sensor to a Collector must be defined

Describing a **Sensor**

- Any **Sensor** has a few generic features
  - ...they can be included in a standard definition of a **Sensor**
  - When the sensor operates
  - How frequently the sensor produces a new measurement
  - They are **timing** attributes
- Other features are specific for the provider
  - ...they are defined as **mixins** for the sensor
  - How data are aggregated (low pass, patterns etc.)
  - How data are published (archive, email, streaming etc.)
- There is no limit to the semantic of the mixins
  - however the hooks to connect a Sensor to a Collector must be defined

A **Collector**

- Represents a flow of measurements between a OCCI **Resource** and a **Sensor**
  - ... yes, the source can be a Sensor in its turn
- The provider has control on the available measurements
- The user has control on the selection and the configuration of the **Collectors**
- Cross provider measurements can be implemented

A **Collector**

- Represents a flow of measurements between a OCCI **Resource** and a **Sensor**
  - ... yes, the source can be a Sensor in its turn
- The provider has control on the available measurements
- The user has control on the selection and the configuration of the **Collectors**
- Cross provider measurements can be implemented

A **Collector**

- Represents a flow of measurements between a OCCI **Resource** and a **Sensor**
  - ... yes, the source can be a Sensor in its turn
- The provider has control on the available measurements
- The user has control on the selection and the configuration of the **Collectors**
- Cross provider measurements can be implemented
  - ... to accomodate the utilization of several providers with a unique dashboard Sensor

A **Collector**

- Represents a flow of measurements between a OCCI **Resource** and a **Sensor**
  - ... yes, the source can be a Sensor in its turn
- The provider has control on the available measurements
- The user has control on the selection and the configuration of the **Collectors**
- Cross provider measurements can be implemented
  - ... to accomodate the utilization of several providers with a unique dashboard Sensor

A **Collector**

- Represents a flow of measurements between a OCCI **Resource** and a **Sensor**
  - ... yes, the source can be a Sensor in its turn
- The provider has control on the available measurements
- The user has control on the selection and the configuration of the **Collectors**
- Cross provider measurements can be implemented
  - ... to accomodate the utilization of several providers with a unique dashboard Sensor

A **Collector**

- Represents a flow of measurements between a OCCI **Resource** and a **Sensor**
  - ... yes, the source can be a Sensor in its turn
- The provider has control on the available measurements
- The user has control on the selection and the configuration of the **Collectors**
- Cross provider measurements can be implemented
  - ... to accomodate the utilization of several providers with a unique dashboard Sensor

Describing a **Collector**

- As in the case of the Sensor there are generic attributes of a collector:
  - The sampling period
  - The accuracy of the sampling period
  - ... again, just timing
- Other attributes are defined by provider-specific mixins with an arbitrary semantic

Describing a **Collector**

- As in the case of the Sensor there are generic attributes of a collector:
  - The sampling period
  - The accuracy of the sampling period
  - ... again, just timing
- Other attributes are defined by provider-specific mixins with an arbitrary semantic

Describing a **Collector**

- As in the case of the Sensor there are generic attributes of a collector:
  - The sampling period
  - The accuracy of the sampling period
  - ... again, just timing
- Other attributes are defined by provider-specific mixins with an arbitrary semantic
  - ... the metric that is measured (throughput, free space, temperature etc.)

Describing a **Collector**

- As in the case of the Sensor there are generic attributes of a collector:
  - The sampling period
  - The accuracy of the sampling period
  - ... again, just timing
- Other attributes are defined by provider-specific mixins with an arbitrary semantic
  - ...the metric that is measured (throughput, free space, temperature etc.)

Describing a **Collector**

- As in the case of the Sensor there are generic attributes of a collector:
  - The sampling period
  - The accuracy of the sampling period
  - ... again, just timing
- Other attributes are defined by provider-specific mixins with an arbitrary semantic
  - ...the metric that is measured (throughput, free space, temperature etc.)

Describing a **Collector**

- As in the case of the Sensor there are generic attributes of a collector:
  - The sampling period
  - The accuracy of the sampling period
  - ... again, just timing
- Other attributes are defined by provider-specific mixins with an arbitrary semantic
  - ...the metric that is measured (throughput, free space, temperature etc.)

A monitoring-aware **Resource**

- A **Resource** that is the target of a monitoring activity may be explicitely characterized with a **Collector End-Point** mixin
- Such a mixin contains the description of the monitoring modality for that resource
  - for instance, the location of a log file
- The presence of this mixin allows cross-provider interoperability
- As an alternative, the modality can be left implicit (default)

A monitoring-aware **Resource**

- A **Resource** that is the target of a monitoring activity may be explicitely characterized with a **Collector End-Point** mixin
- Such a mixin contains the description of the monitoring modality for that resource
  - for instance, the location of a log file
- The presence of this mixin allows cross-provider interoperability
- As an alternative, the modality can be left implicit (default)

A monitoring-aware **Resource**

- A **Resource** that is the target of a monitoring activity may be explicitely characterized with a **Collector End-Point** mixin
- Such a mixin contains the description of the monitoring modality for that resource
  - for instance, the location of a log file
- The presence of this mixin allows cross-provider interoperability
- As an alternative, the modality can be left implicit (default)

A monitoring-aware **Resource**

- A **Resource** that is the target of a monitoring activity may be explicitely characterized with a **Collector End-Point** mixin
- Such a mixin contains the description of the monitoring modality for that resource
  - for instance, the location of a log file
- The presence of this mixin allows cross-provider interoperability
- As an alternative, the modality can be left implicit (default)

A monitoring-aware **Resource**

- A **Resource** that is the target of a monitoring activity may be explicitely characterized with a **Collector End-Point** mixin
- Such a mixin contains the description of the monitoring modality for that resource
  - for instance, the location of a log file
- The presence of this mixin allows cross-provider interoperability
- As an alternative, the modality can be left implicit (default)

A monitoring-aware **Resource**

- A **Resource** that is the target of a monitoring activity may be explicitely characterized with a **Collector End-Point** mixin
- Such a mixin contains the description of the monitoring modality for that resource
  - for instance, the location of a log file
- The presence of this mixin allows cross-provider interoperability
- As an alternative, the modality can be left implicit (default)

New! This feature is not included in the available revision of the document

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
- The two **Kinds** have a OCCI schema associated
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - *Aggregator* mixins describe the aggregation activity of a *Sensor*
  - *Publisher* mixins describe the rendering activity of a *Sensor*
  - *Metric* mixins describe the measurement activity of a *Collector*
  - *Collector Endpoint* mixins describe the management access point of a target resource
- The two **Kinds** have a OCCI schema associated
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the management access point of a target resource
- The two **Kinds** have a OCCI schema associated
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
  - they are defined in the standard
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
  - ...they are defined in the standard
- The **Mixins** may be associated with a provider specific schema

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
  - ...they are defined in the standard
- The **Mixins** may be associated with a provider specific schema
  - ...but we do not exclude that some of them may be part of another standard

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
  - ...they are defined in the standard
- The **Mixins** may be associated with a provider specific schema
  - ...but we do not exclude that some of them may be part of another standard

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
  - ...they are defined in the standard
- The **Mixins** may be associated with a provider specific schema
  - ...but we do not exclude that some of them may be part of another standard

The overall picture

- Two entity kinds
  - **Sensor** aggregates and delivers measurements
  - **Collector** acquires measurements
- Four mixin types
  - **Aggregator** mixins describe the aggregation activity of a *Sensor*
  - **Publisher** mixins describe the rendering activity of a *Sensor*
  - **Metric** mixins describe the measurement activity of a *Collector*
  - **Collector Endpoint** mixins describe the monitoring access point of a target resource
- The two **Kinds** have a OCCI schema associated
  - ...they are defined in the standard
- The **Mixins** may be associated with a provider specific schema
  - ...but we do not exclude that some of them may be part of another standard

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - Input attributes
  - Output attributes
- **Input** and **Output** hooks with matching labels are connected
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
  - this may mean a data flow among them
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
  - this may mean a data flow among them
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
  - this may mean a data flow among them
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
  - this may mean a data flow among them
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
  - this may mean a data flow among them
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Hold them together: input and output hooks

- The designer needs a tool to assemble a monitoring infrastructure
  - we introduce input and output attributes for the *Mixins*
- We introduce **hook attributes** (*named* attributes in the last revision) for a mixin:
  - their value corresponds to a label
  - **Input** attributes
  - **Output** attributes
- **Input** and **Output** hooks with matching labels are connected
  - this may mean a data flow among them
- The scope of hook labels is limited to a sensor and its adjacent collectors
- The provider indicates hook semantics in the specifications of the **mixin**

Step by step design of a monitoring infrastructure

An example:

- One sensor collects measurementes from two resources
- Results are published through two different channels (e.g., streaming and database)
- Two distinct measurement tools are applied to each of the two resources (total four tools)
- We combine a metric from both resources (e.g., average load)

Step by step design of a monitoring infrastructure

An example:

- One sensor collects measurementes from two resources
- Results are published through two different channels (e.g., streaming and database)
- Two distinct measurement tools are applied to each of the two resources (total four tools)
- We combine a metric from both resources (e.g., average load)

Step by step design of a monitoring infrastructure

An example:

- One sensor collects measurementes from two resources
- Results are published through two different channels (e.g., streaming and database)
- Two distinct measurement tools are applied to each of the two resources (total four tools)
- We combine a metric from both resources (e.g., average load)

Step by step design of a monitoring infrastructure

An example:

- One sensor collects measurementes from two resources
- Results are published through two different channels (e.g., streaming and database)
- Two distinct measurement tools are applied to each of the two resources (total four tools)
- We combine a metric from both resources (e.g., average load)

Step by step design of a monitoring infrastructure

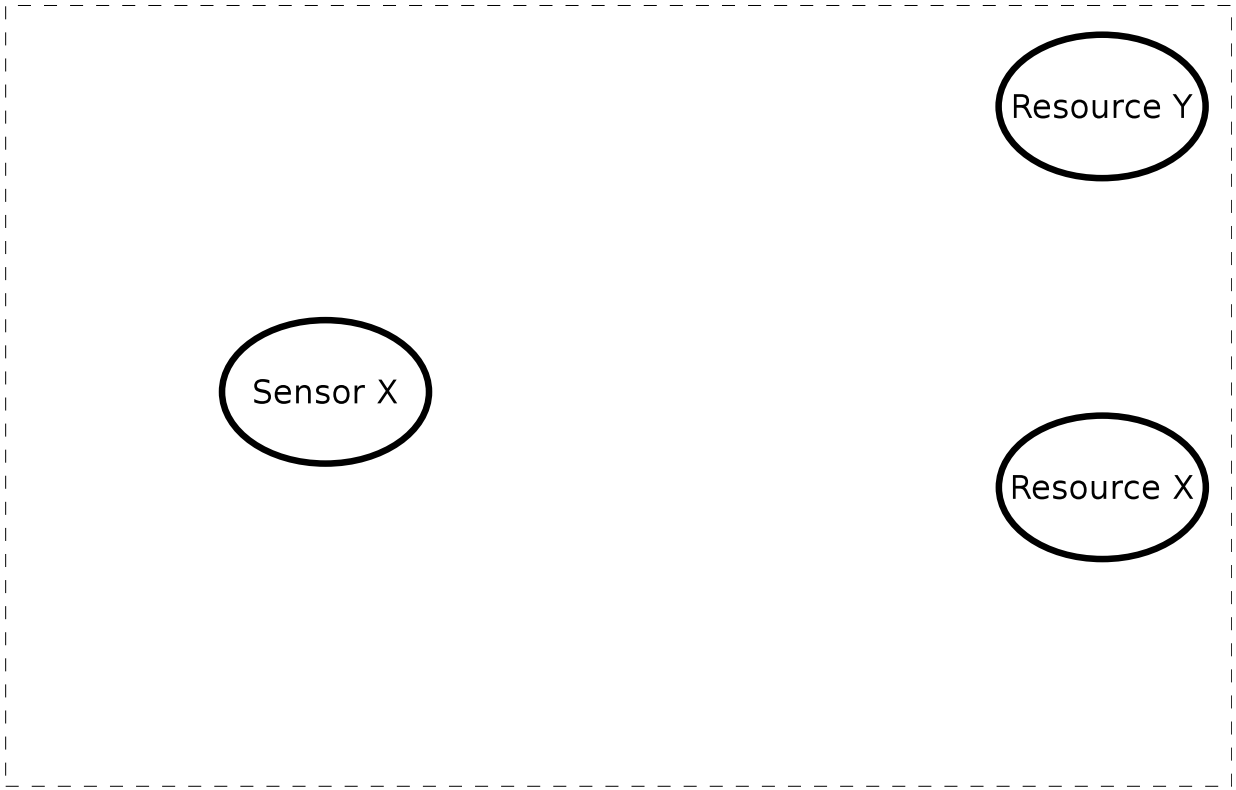
An example:

- One sensor collects measurementes from two resources
- Results are published through two different channels (e.g., streaming and database)
- Two distinct measurement tools are applied to each of the two resources (total four tools)
- We combine a metric from both resources (e.g., average load)

Step by step design of a monitoring infrastructure

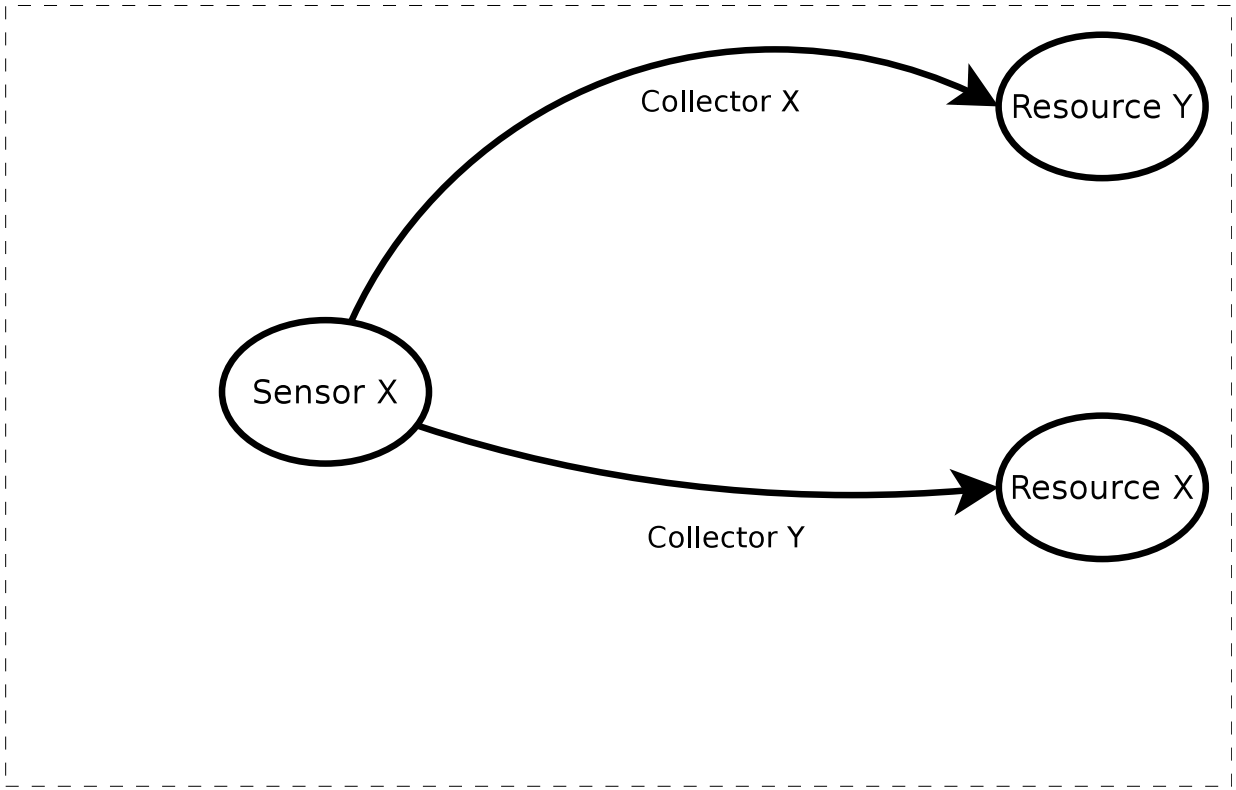
The resources we want to monitor: they have a **Collector Endpoint** mixin associated

Step by step design of a monitoring infrastructure



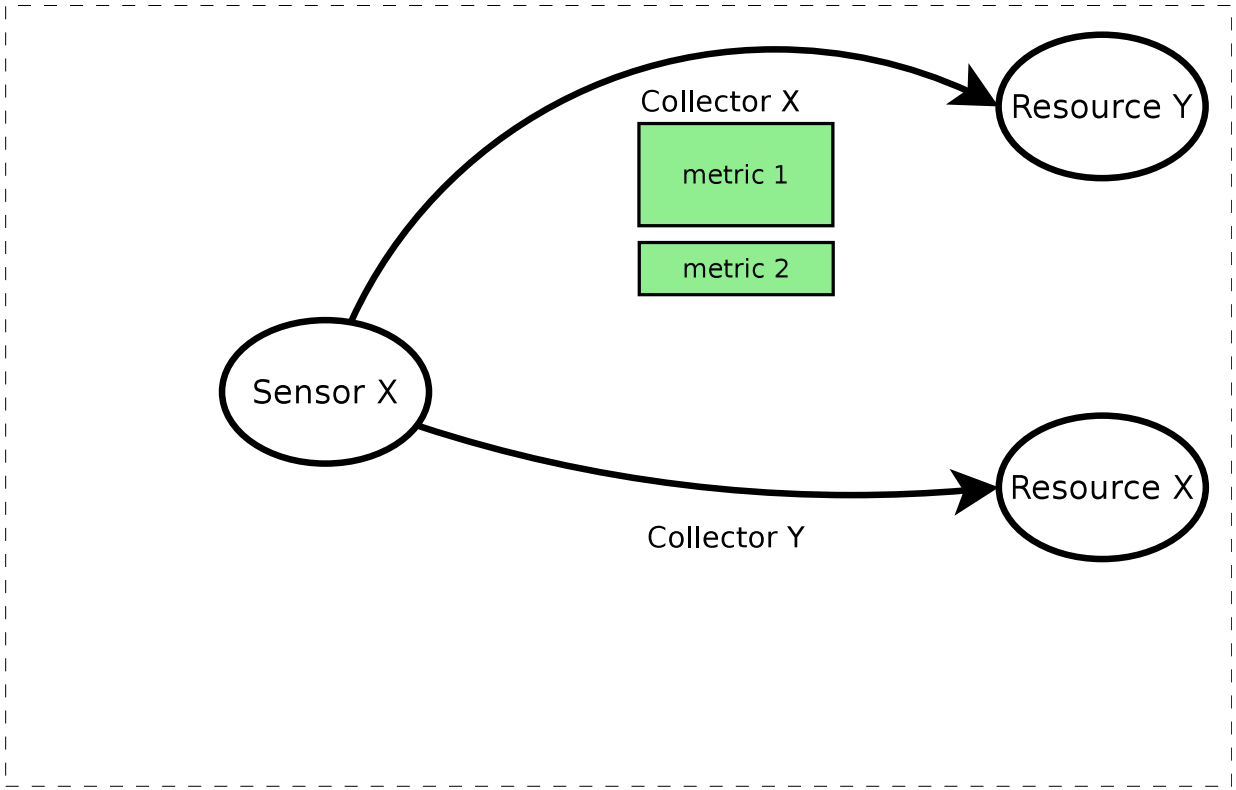
Create one Sensor resource

Step by step design of a monitoring infrastructure



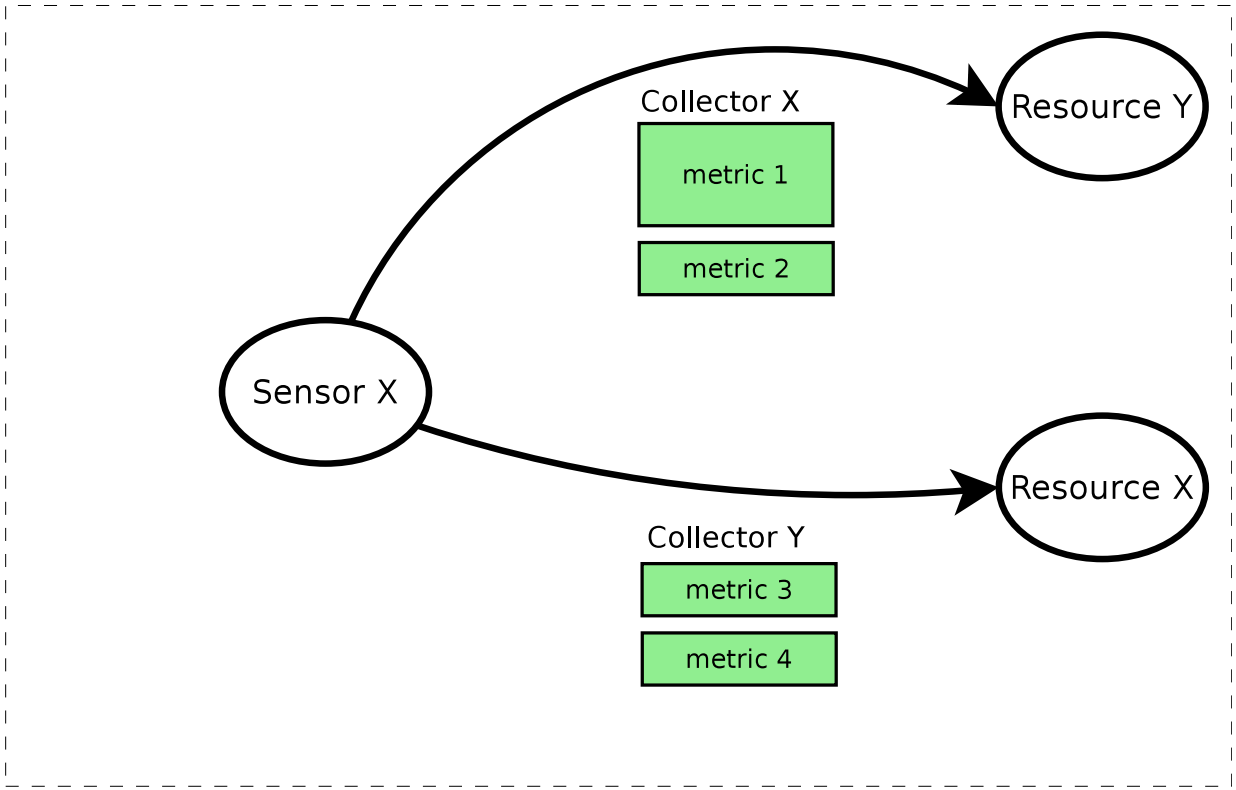
Use two collectors to define the measurement activiy

Step by step design of a monitoring infrastructure



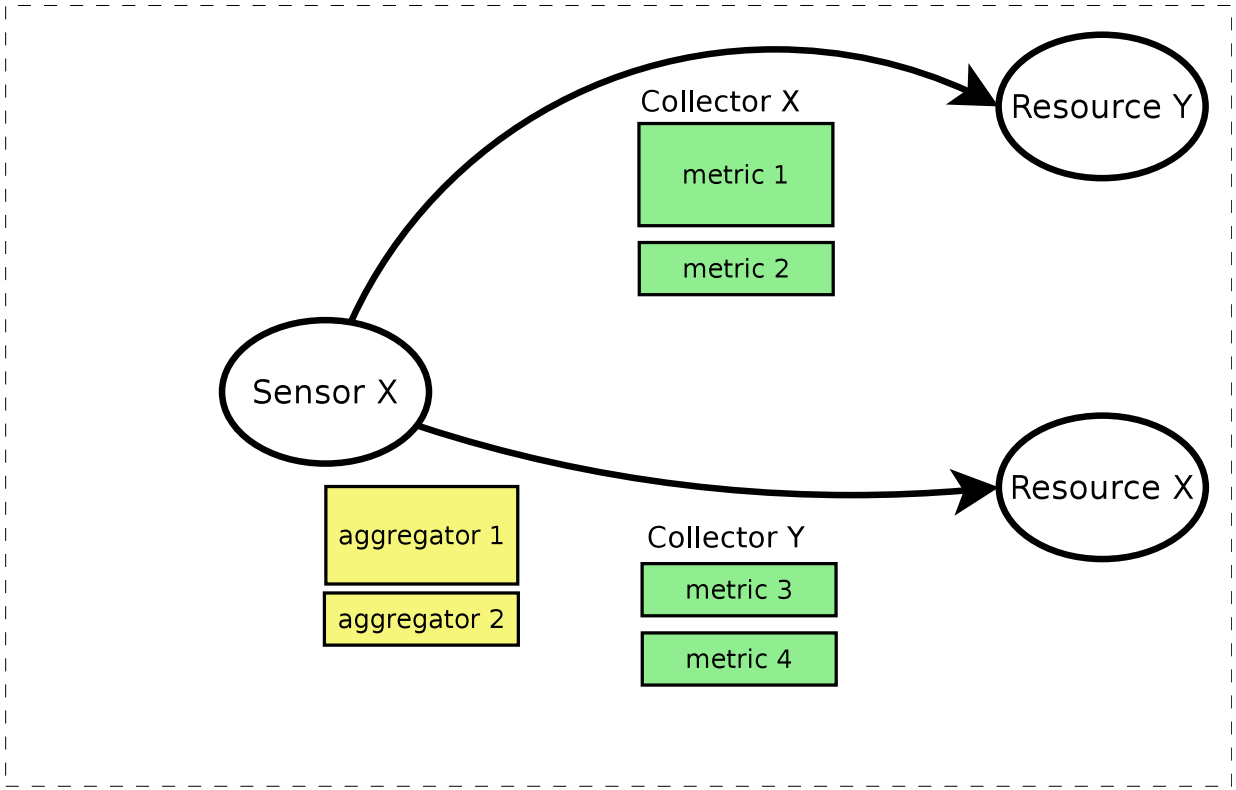
Associate two metric mixins to the Collector X

Step by step design of a monitoring infrastructure



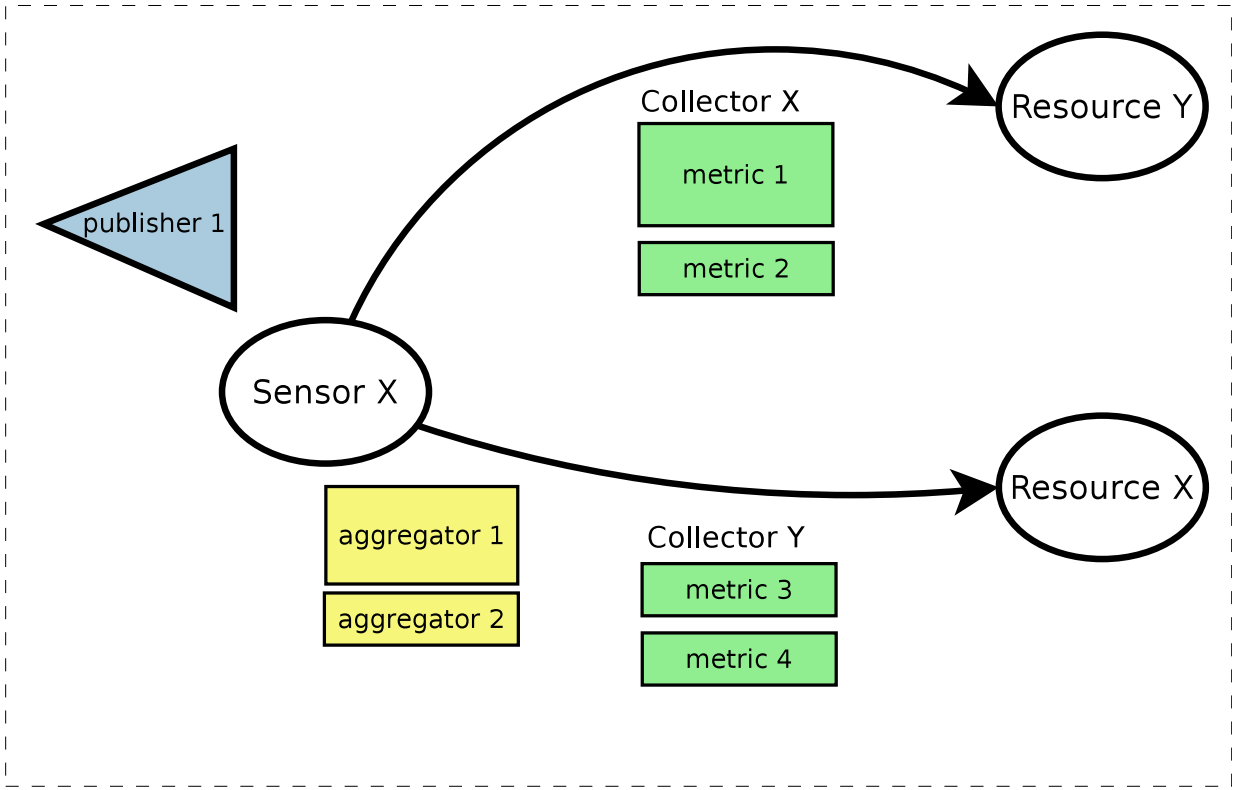
And another two metric mixins to the Collector Y

Step by step design of a monitoring infrastructure



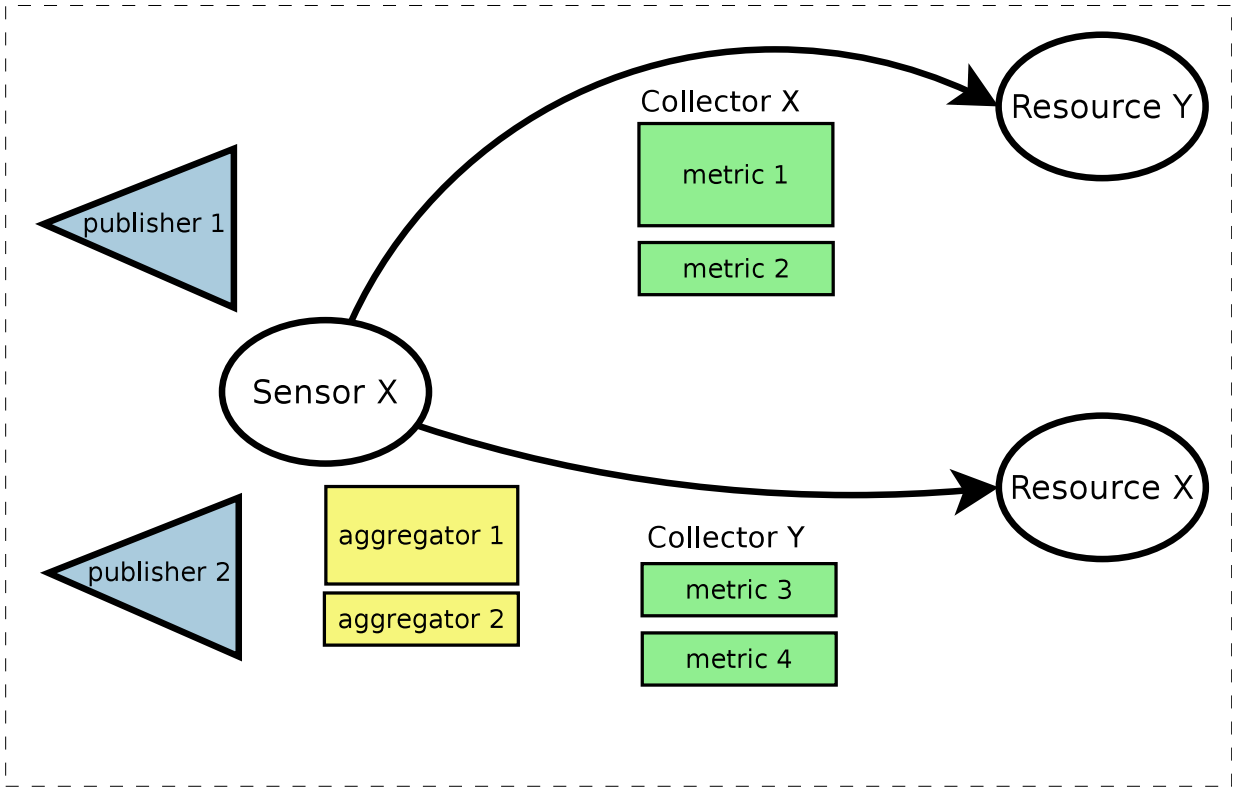
Associate two aggregator mixins to the Sensor

Step by step design of a monitoring infrastructure



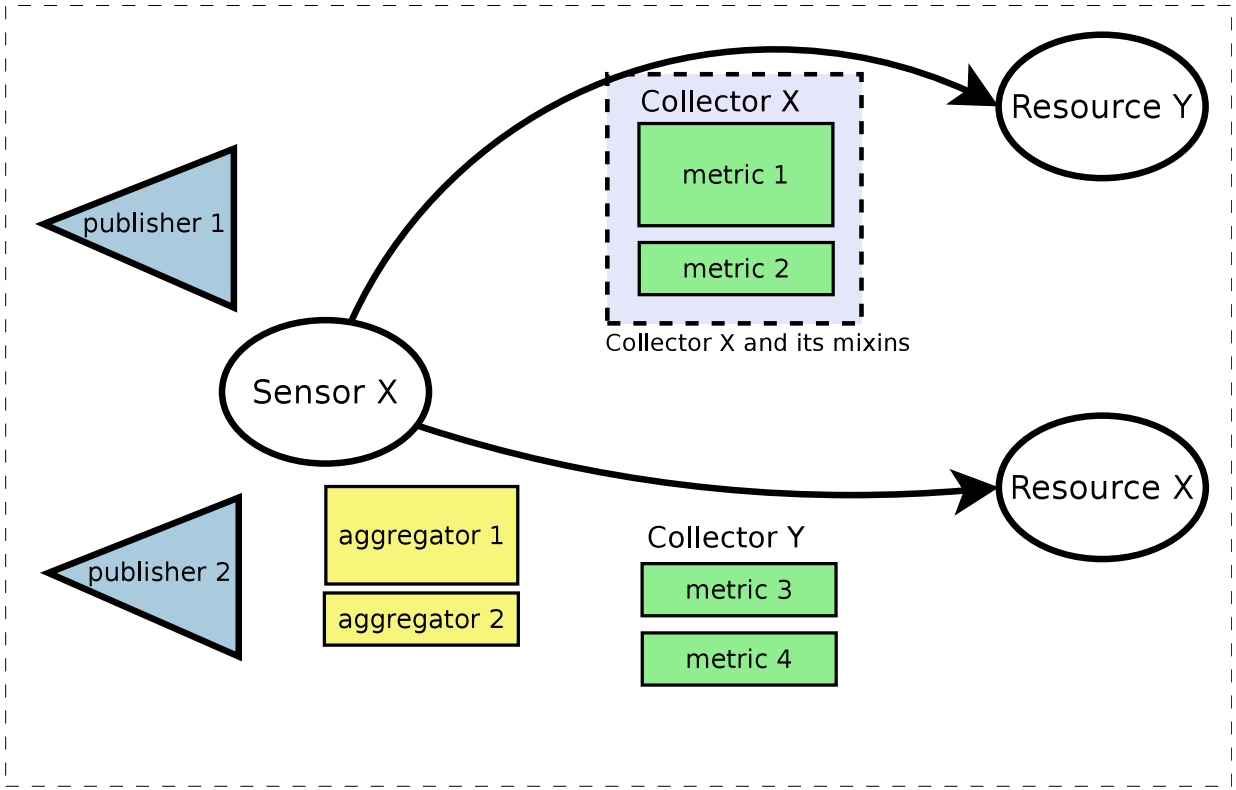
One publisher is going to use raw data from the collector

Step by step design of a monitoring infrastructure



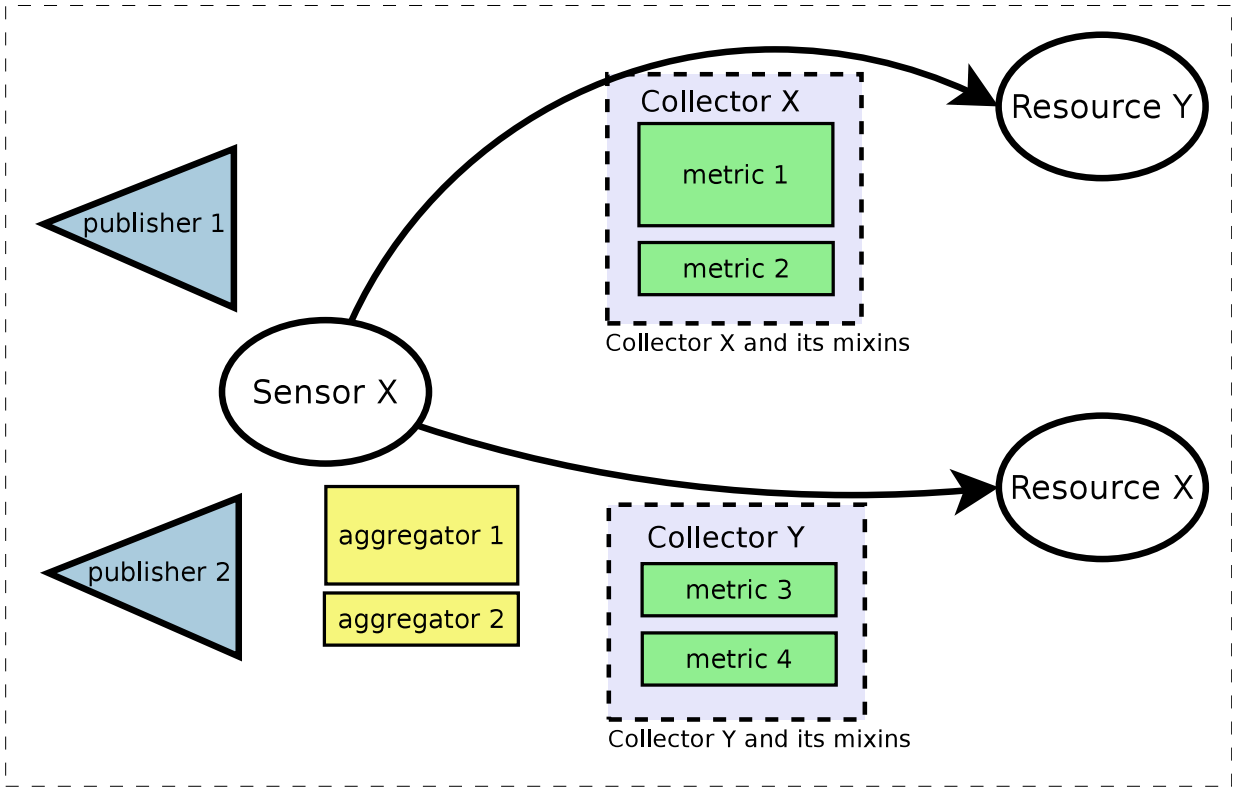
Another is going to receive measurements from the aggregators

Step by step design of a monitoring infrastructure



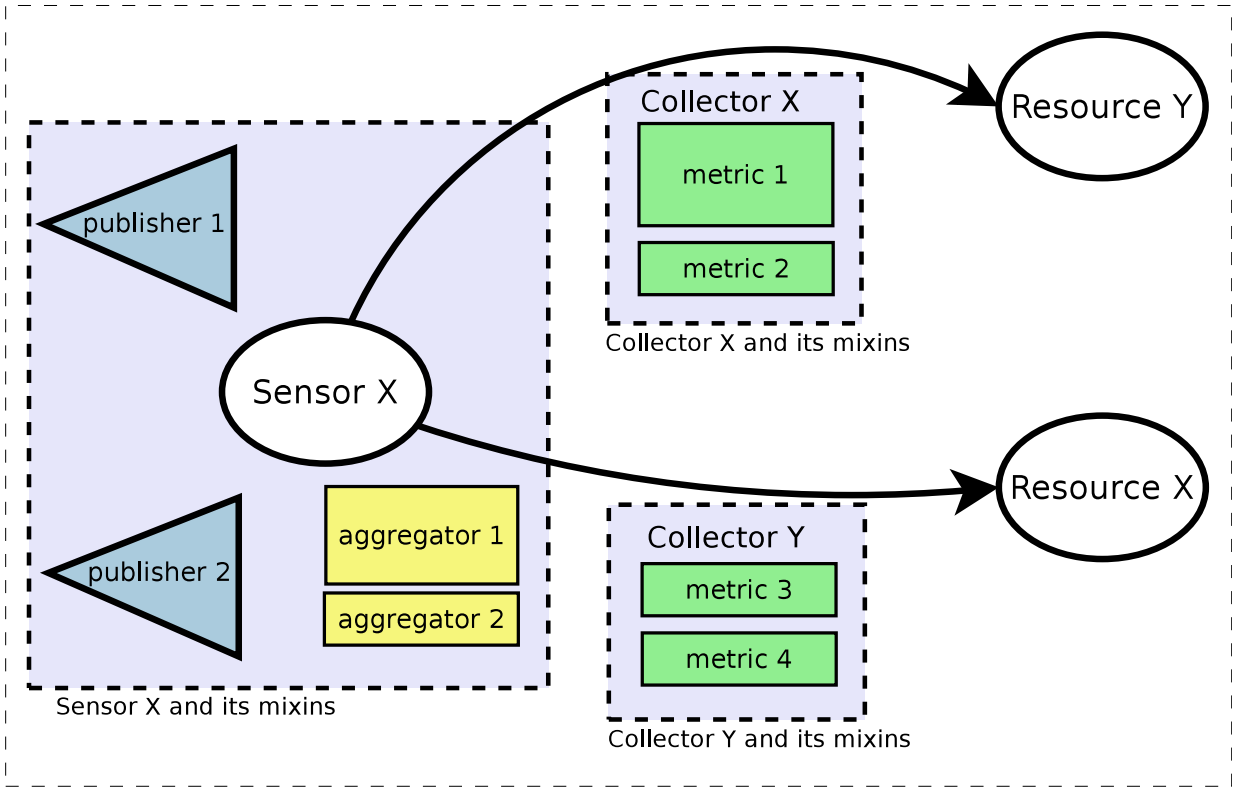
A frame for Collector X and its mixins

Step by step design of a monitoring infrastructure



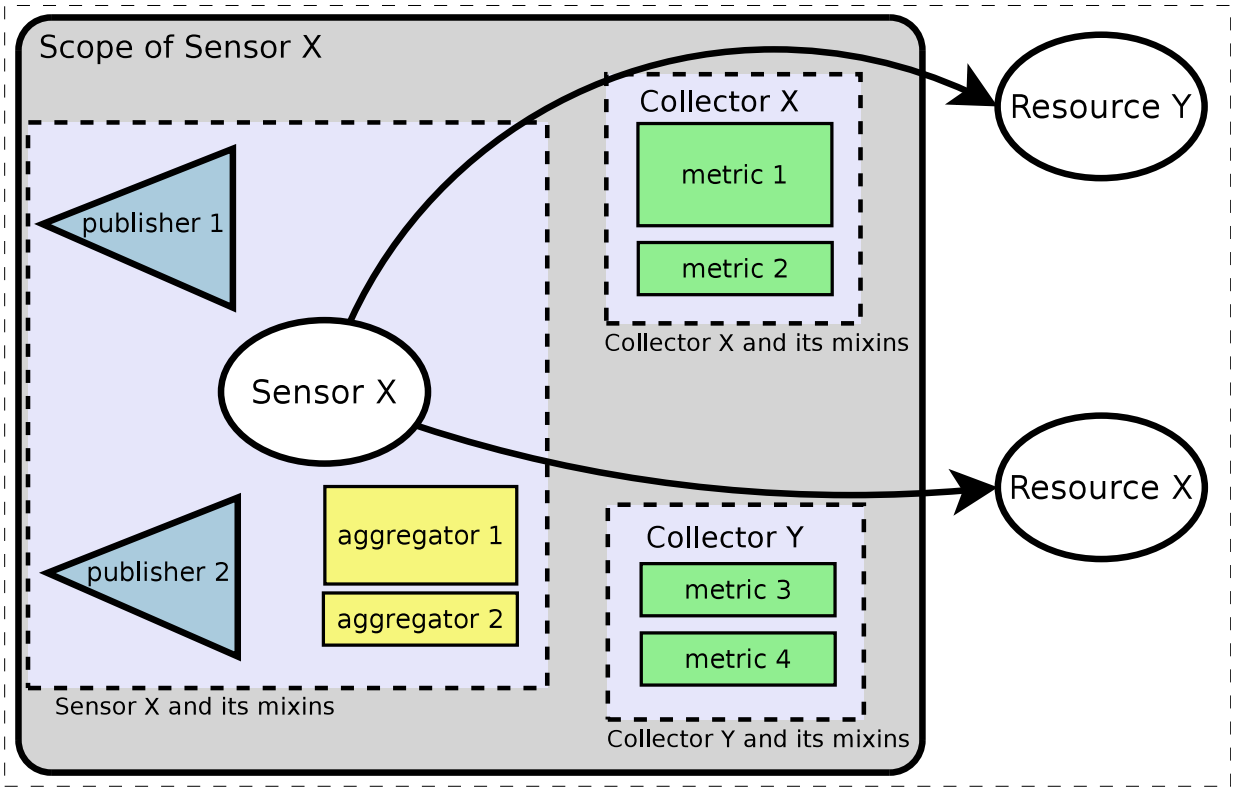
... one for Collector Y...

Step by step design of a monitoring infrastructure



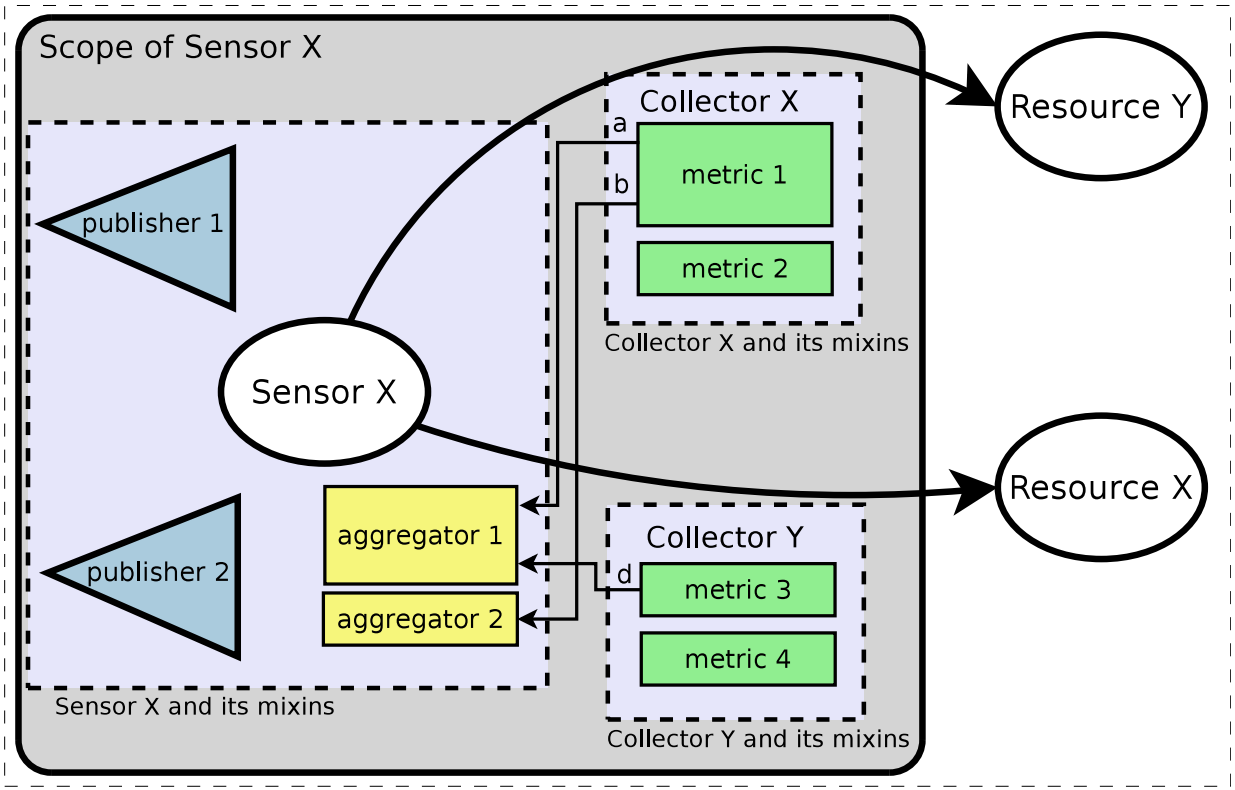
... one for the sensor

Step by step design of a monitoring infrastructure



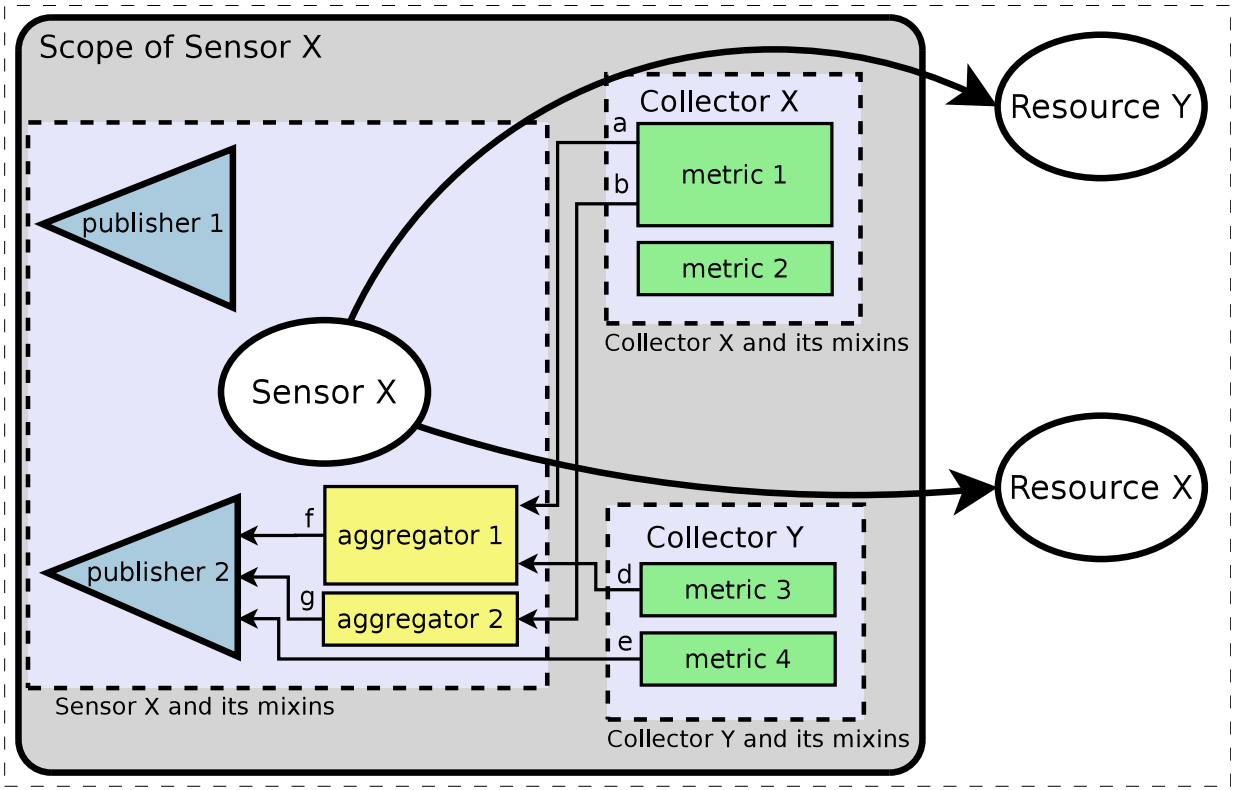
The scope of the Sensor (for hook labels)

Step by step design of a monitoring infrastructure



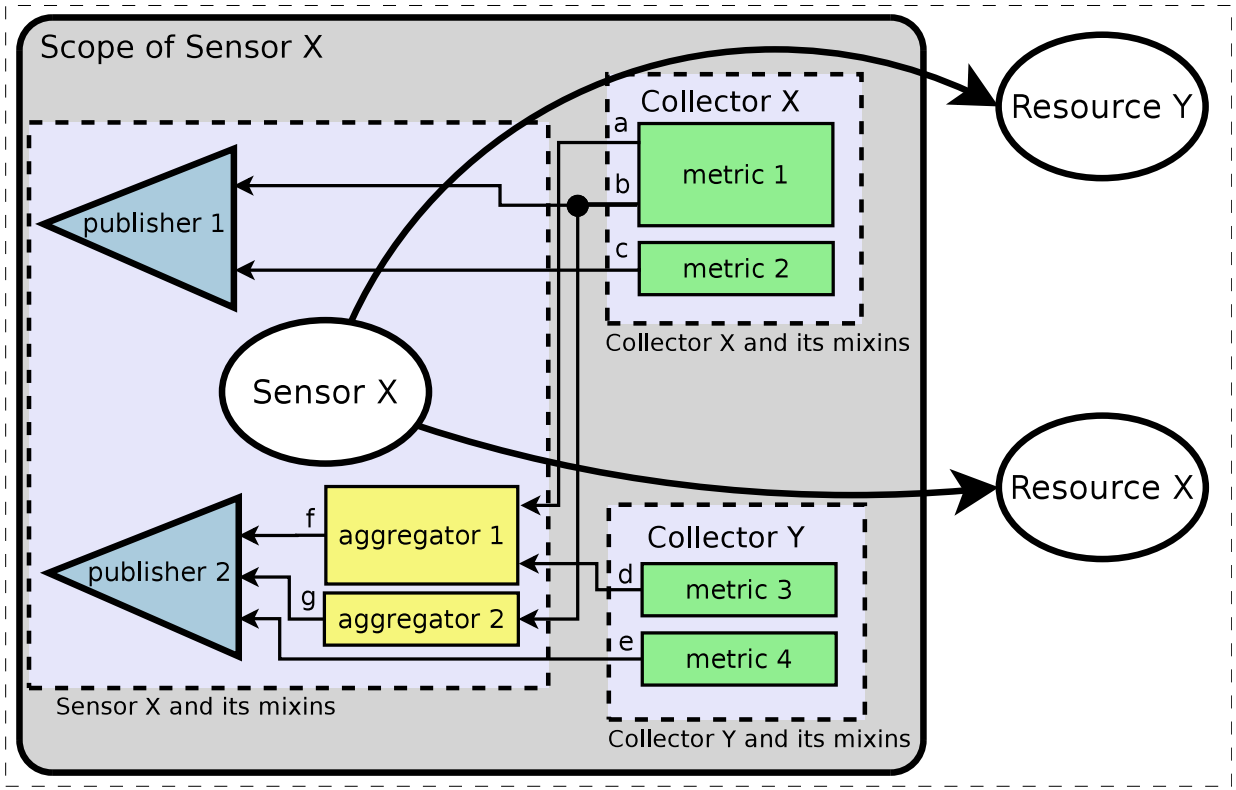
Feeding the aggregators: a,b,d are hook labels

Step by step design of a monitoring infrastructure



Feeding publisher 2: aggregated (f,g) and raw data (e)

Step by step design of a monitoring infrastructure



Feeding publisher 1: measurement stream b is multicast

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - Collector link to produce monitoring data
  - Sensor resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

Conclusions

- To manage cloud resource, we need to monitor them
  - indeed many providers offer monitoring as a service
- We establish a minimum common ground for interoperability
  - a standard, aligned with the Open Cloud Computing Interface
- Two entities:
  - **Collector** link to produce monitoring data
  - **Sensor** resource to aggregate and deliver monitoring data
- Finalized using mixins defined by the provider
- Can be combined to form complex monitoring infrastructures
- More... may be extended to **any** computational infrastructure

End of part 1

From the OCCI to Java: unfolding the infrastructure

A Java backend for OCCI monitoring

Augusto Ciuffoletti

Dept. of Computer Science - Univ. of Pisa

September 5, 2014

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

An experiment to verify an abstract specification

The framework

- Limited to the backend, in the perspective of the provider that implements the monitoring service
- No user interface: OCCI-JSON documents are already on the web server
- Written in Java, because it is widely understood

The question - Is the specification realistic?

- is the sensor+collector model sufficiently descriptive?
- are the attributes enough to describe a monitoring framework?
- are mixins modular with respect to the framework?
- is there a simple way to implement it efficiently?

The answers are basically positive, but something interesting has been discovered...

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

Basic design options

- The **Collector Endpoint** mixin is implemented on a **compute** resource as a daemon with an RMI interface
- **Metric** mixins are dynamically created threads (Java reflection)
- Metric classes are not dynamically loaded (todo)
- The **Sensor** runs on a distinct host, possibly shared with other Sensors
- Data flow from the *Collector Endpoint* to the **Sensor** uses a TCP channel with JSON encoding
- **Aggregator** and **Publisher** mixins are dynamically created threads (as for **Metrics**)
- communication between **Aggregators** and **Publishers** uses internal pipes and JSON
- input and output hook attributes are respectively associated with *PipedInputStreams* and *PrintWriters*

The tools

- Java 7: either Oracle or OpenJDK
- json-simple 1.1.1

  JSON.simple is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode JSON text.

- jsoup 1.7.3

The tools

- Java 7: either Oracle or OpenJDK
- json-simple 1.1.1

  JSON.simple is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode JSON text.

- jsoup 1.7.3

  jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of HTML, CSS, and jquery-like methods.

The tools

- Java 7: either Oracle or OpenJDK
- json-simple 1.1.1

  JSON.simple is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode JSON text.

- jsoup 1.7.3

  jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

The tools

- Java 7: either Oracle or OpenJDK
- json-simple 1.1.1

  JSON.simple is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode JSON text.

- jsoup 1.7.3

  jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

The tools

- Java 7: either Oracle or OpenJDK
- json-simple 1.1.1

  JSON.simple is a simple Java toolkit for JSON. You can use JSON.simple to encode or decode JSON text.

- jsoup 1.7.3

  jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

An example: monitoring load and connectivity of a **Compute** resource

- The measurement of the average CPU load is sent outside the cloud as a UDP flow
- The connectivity with the gateway is collected in a log file on the sensor.

An example: The Collector


```
{
  "id": "urn:uuid:2345",
  "kind": "http://schemas.ogf.org/occi/monitoring#collector",
  "mixins": [
    "http://example.com/occi/monitoring/metric#CPUPercent"
    "http://example.com/occi/monitoring/metric#isReachable"
  ],
  "attributes": {"occi": {"collector": {
    "period": 60
  }},
  "com": {"example": {"occi": { "monitoring": {
    "CPUPercent" : {"out": "a"},
    "IsReachable" : {"hostname": "192.168.5.1","maxdelay": 1000,
                     "out":"b"}
  }}}}},
  "actions": [],
  "target":"urn:uuid:s1111",
  "source":"urn:uuid:c2222"
}
```

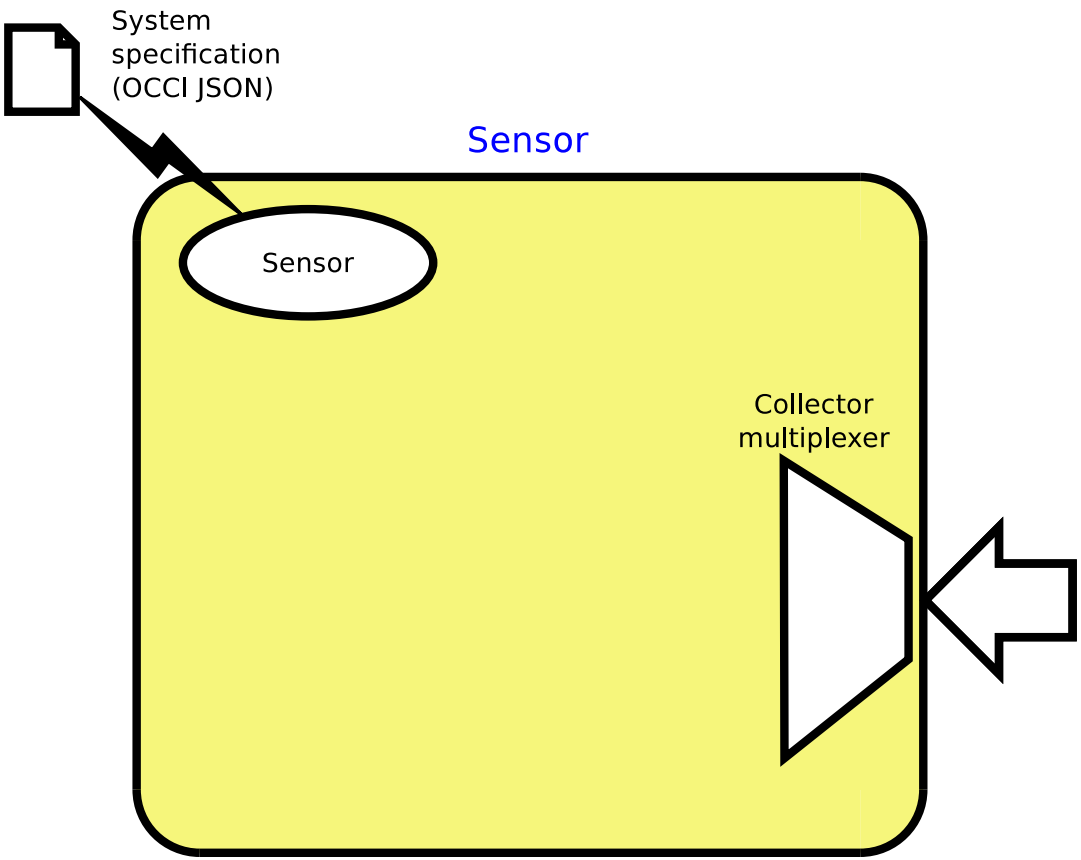
An example: The Sensor


```
{
  "id": "urn:uuid:s1111",
  "kind": "http://schemas.ogf.org/occi/monitoring#sensor",
  "mixins": [
    "http://example.com/occi/monitoring/publisher#SendUDP",
    "http://example.com/occi/monitoring/aggregator#EWMA"
    "http://example.com/occi/monitoring/publisher#Log
  ],
  "attributes": { "occi": { "sensor": {
    "period": 60, "timebase": 1386925386,
    "timestart": 600, "timestop": 3600,
    "networkInterface": "eth0"}
  },
  "com": {"example": {"occi": {"monitoring": {
    "SendUDP" : {"udpAddr": "192.168.5.2:8888","input": "c"},
    "EWMA" : {"gain": 16,"instream": "a","outstream": "c"},
    "Log" : {"filename": "my/log/file","in_msg": "b"}
  }}}}},
  "links": ["urn:uuid:2345"]
}
```

An example: The target Resource

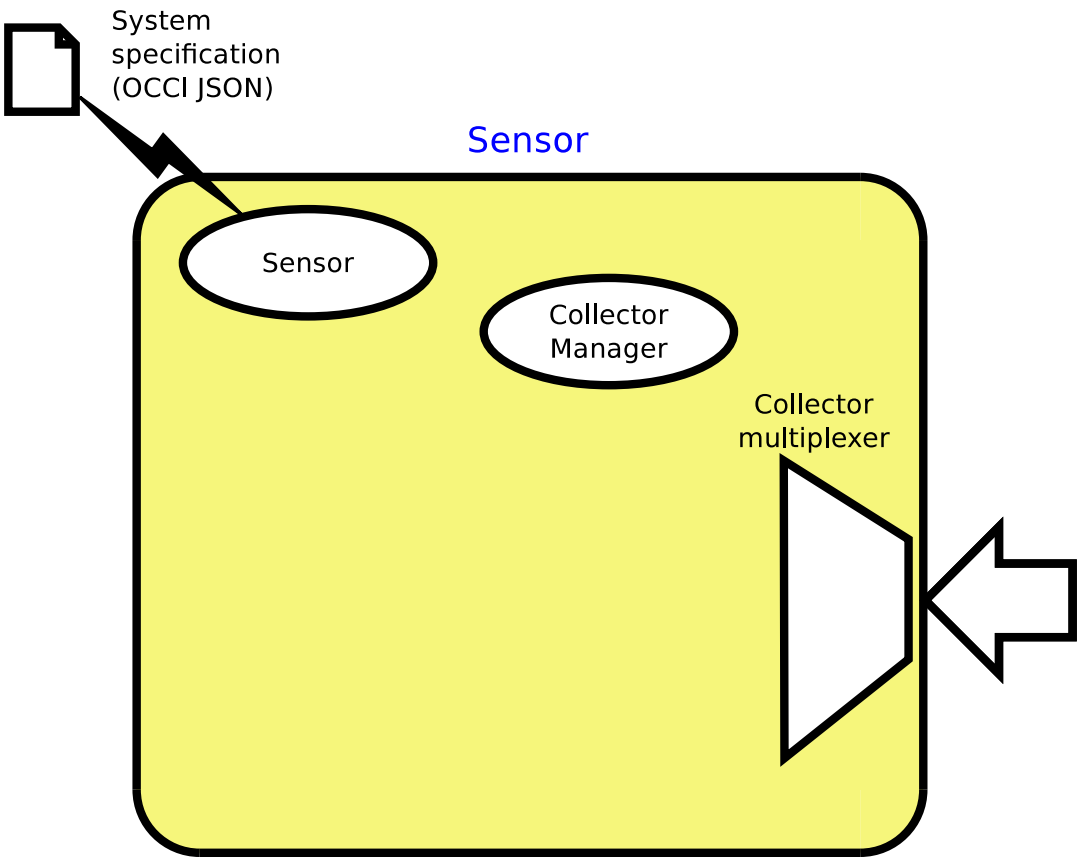

```
{
  "id": "urn:uuid:c2222",
  "kind": "http://schemas.ogf.org/occi/infrastructure#compute",
  "mixins": [
     "http.//schemas.ogf.org/infrastructure/compute#RMIMetricContainer"
  ],
  "attributes": { "occi": { "compute":
    { "speed": 2, "memory": 4, "cores": 2,
      "MetricContainerIPAddress": "192.168.5.3",
      "MetricContainerPort": 12312 }}},
  "actions": []
}
```

Unfolding the sensor



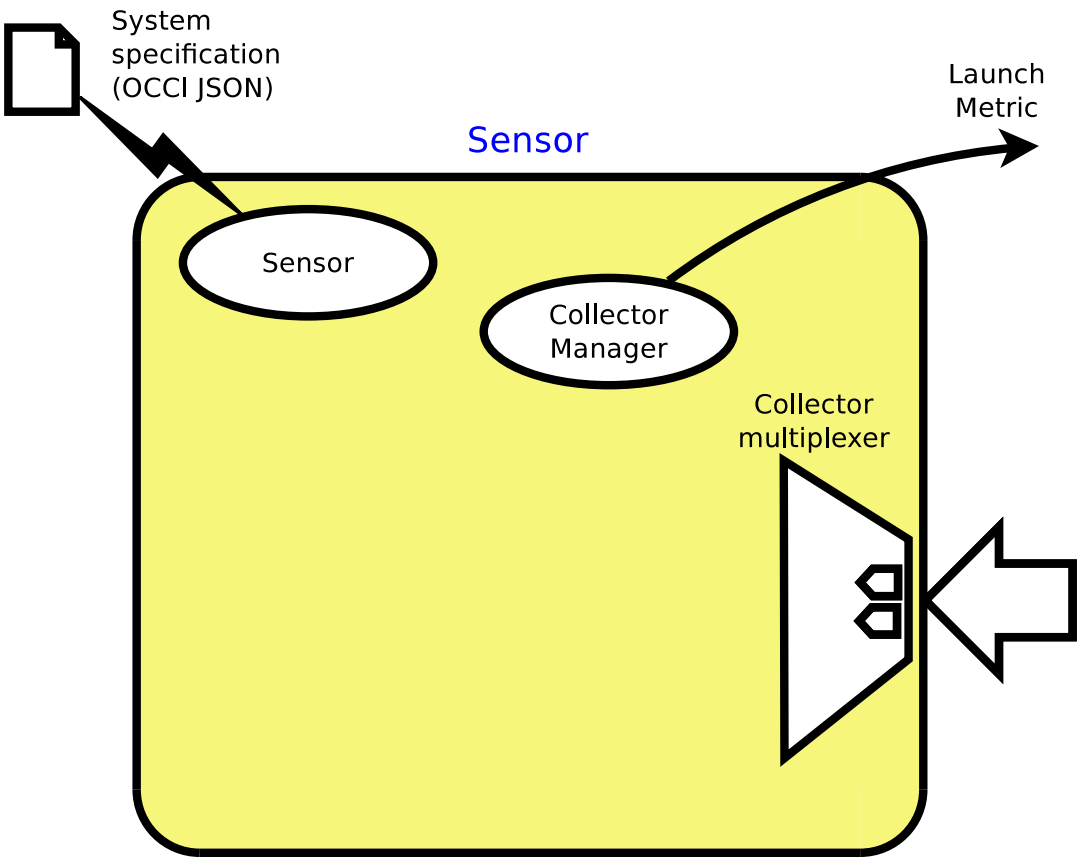
- The sensor starts as a tread in a thread pool
- It reads its specifications using an HTTP GET
- A TCP socket (server side) is allocated for input from the collectors

Unfolding the sensor



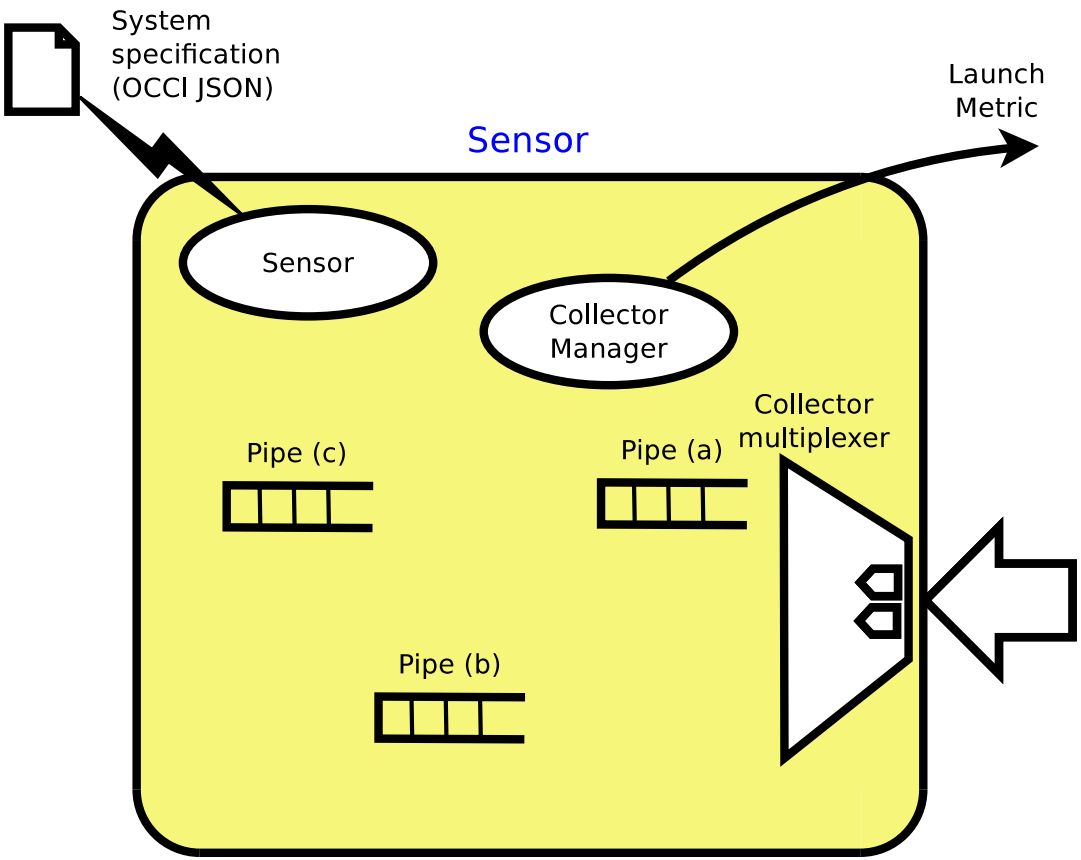
- The sensor launches a *Collector Manager* thread for each collector in the scope

Unfolding the sensor



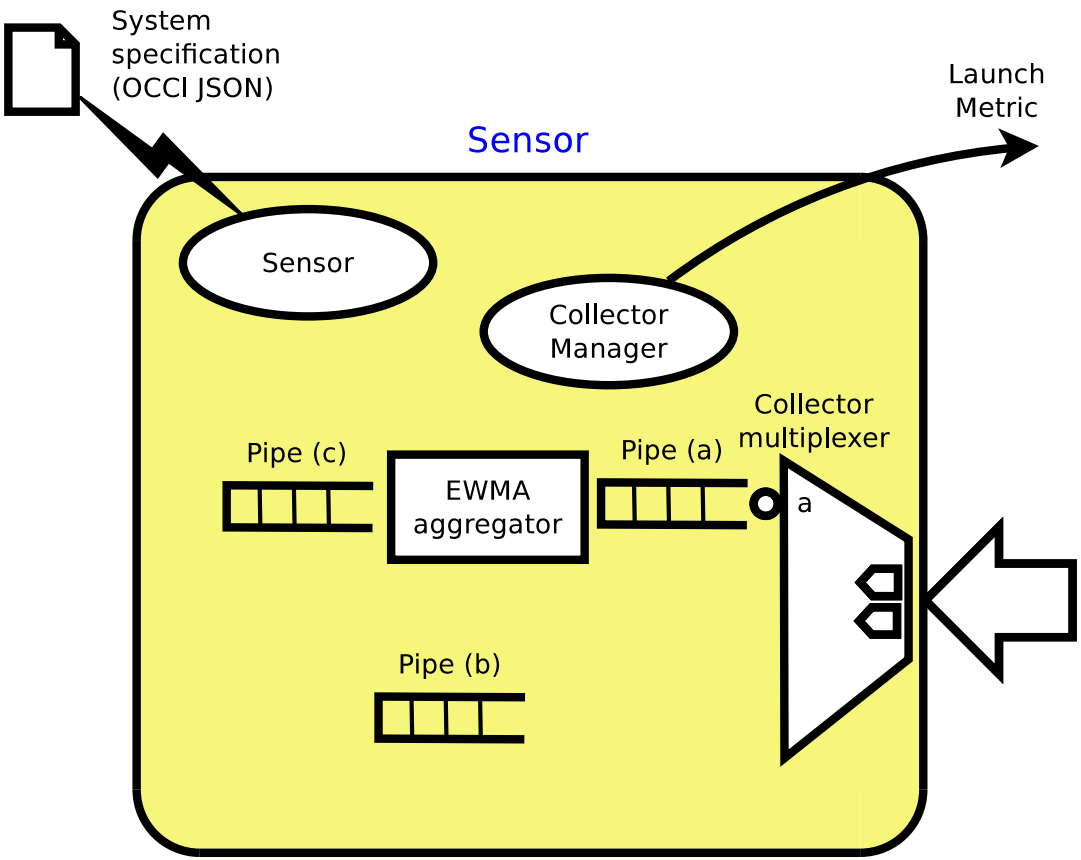
- The Collector Manager invokes (by RMI) the measurement threads on the source
- Each of them opens a TCP connection with the sensor socket

Unfolding the sensor



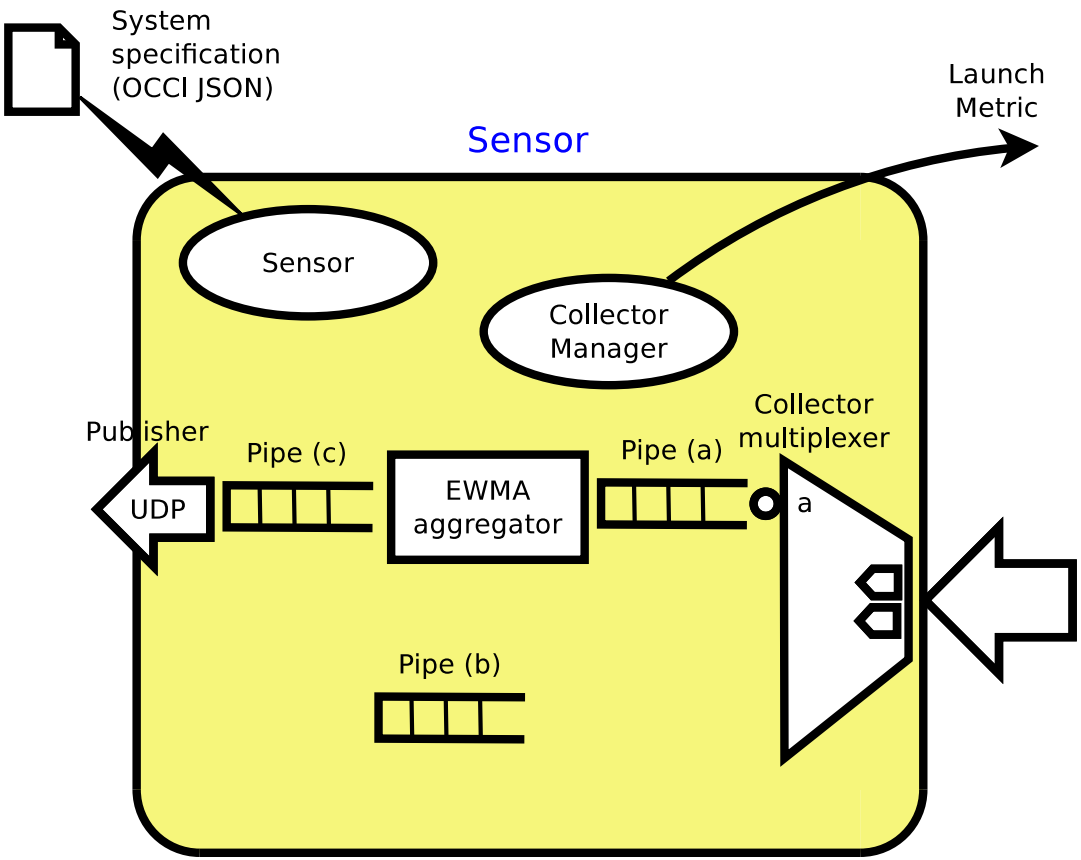
- The Collector Manager creates the pipes used for communication
- There is one pipe for each *hook label* in the scope
- Records from the socket are multiplexed to pipes according with hook identifiers

Unfolding the sensor



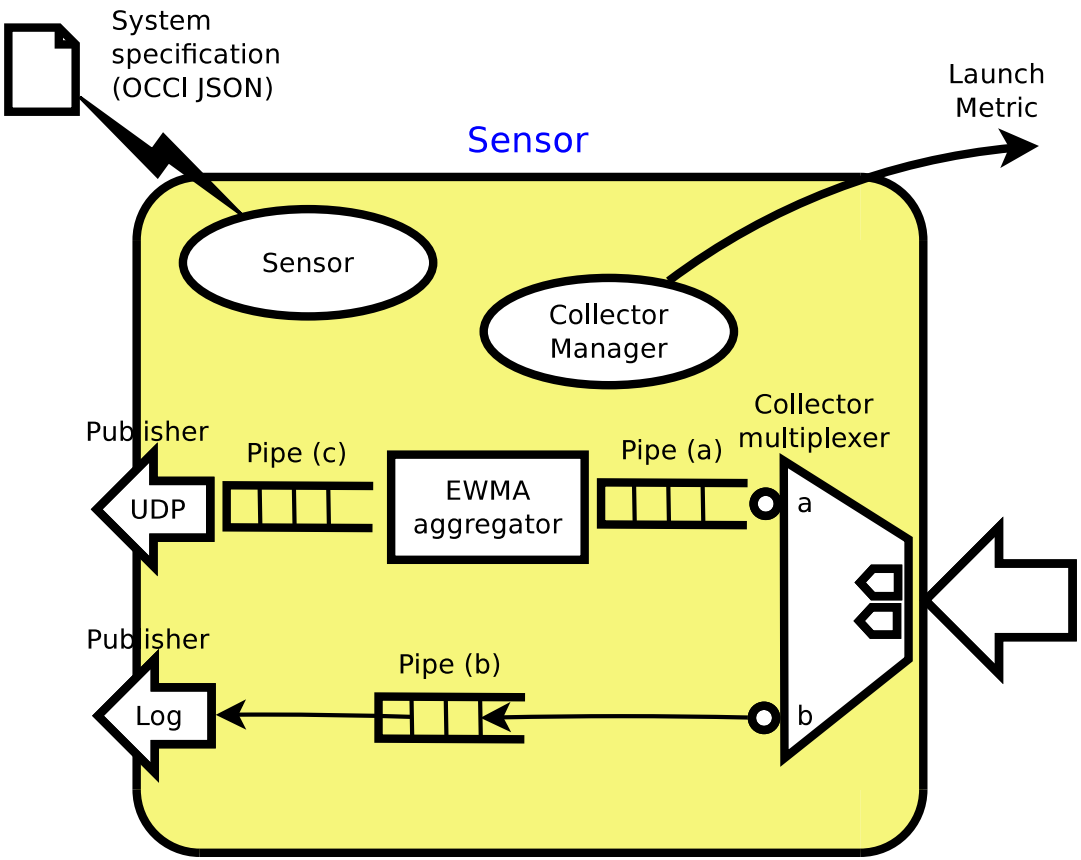
- Create the threads that implement the Aggregators
- Input and output hooks are bound to pipes

Unfolding the sensor



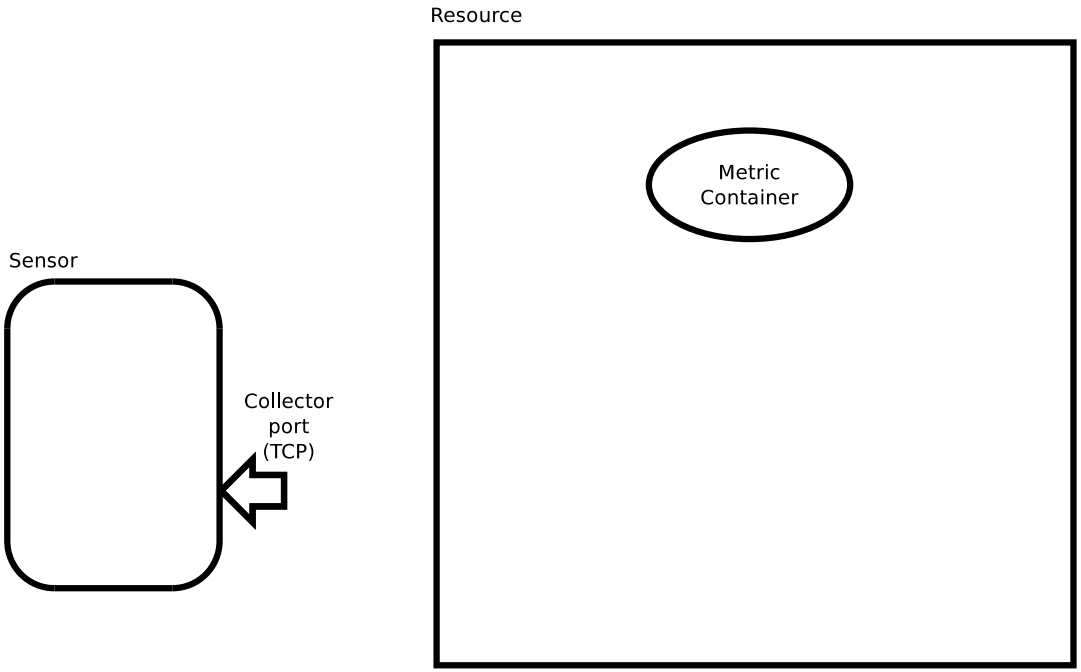
- Create the threads that implement the Publishers
- Input and output channels are bound as for Aggregators

Unfolding the sensor



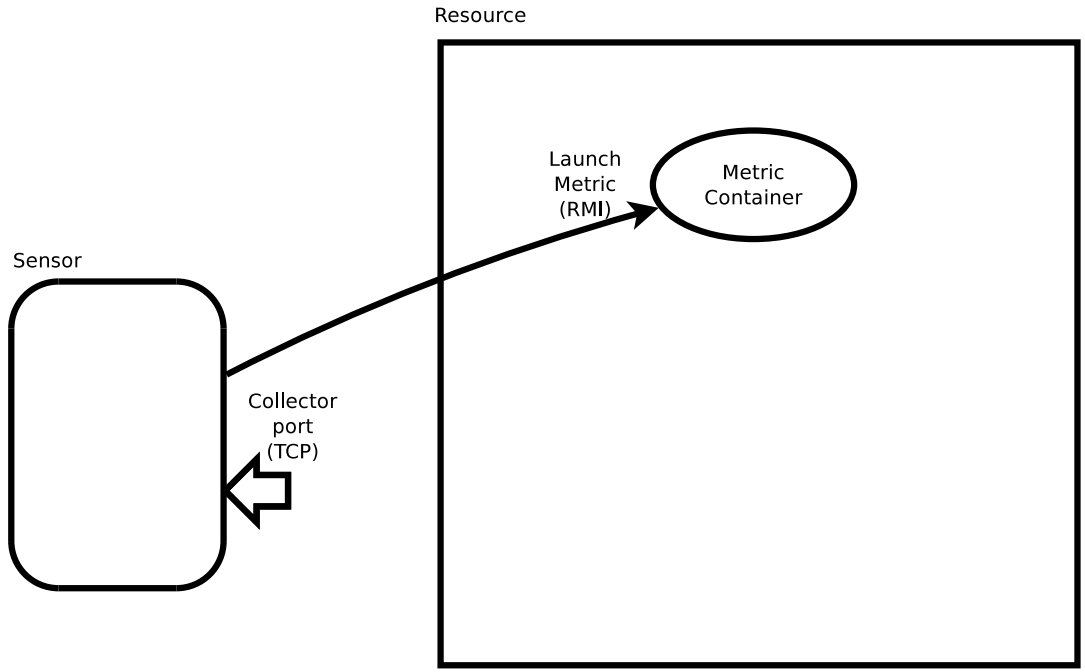
- One publisher is going to use raw data from the collector

Unfolding the Collector Endpoint



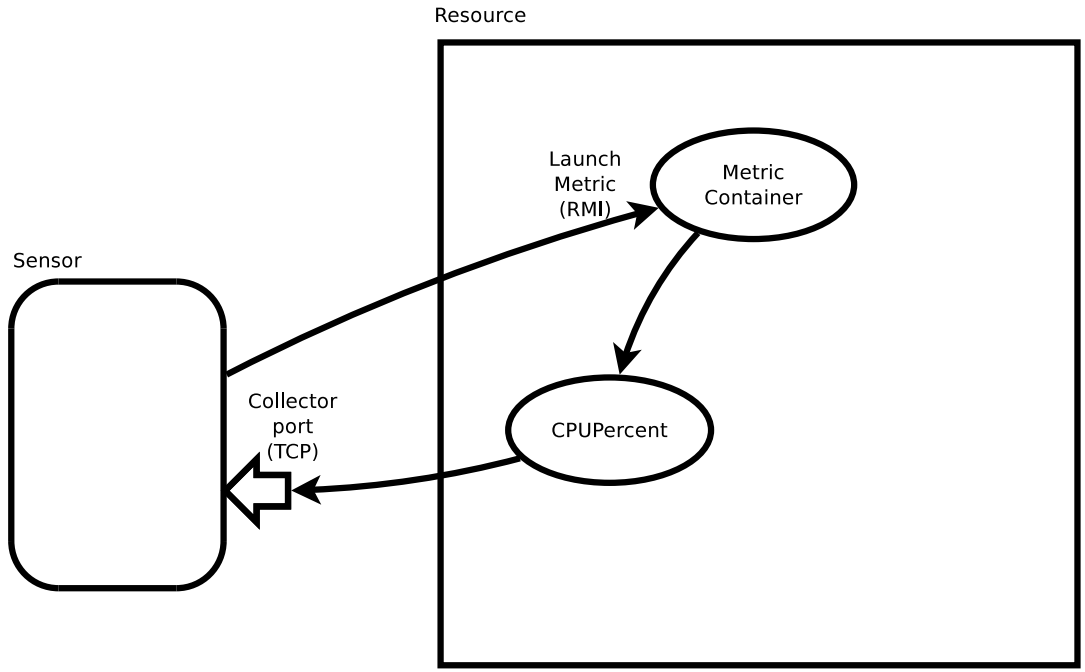
- The monitored resource runs a daemon with an RMI interface (MetricContainer)
- The sensor has a TCP server-side port accepting connections
- The sensor learns the RMI interface from OCCI documents

Unfolding the Collector Endpoint



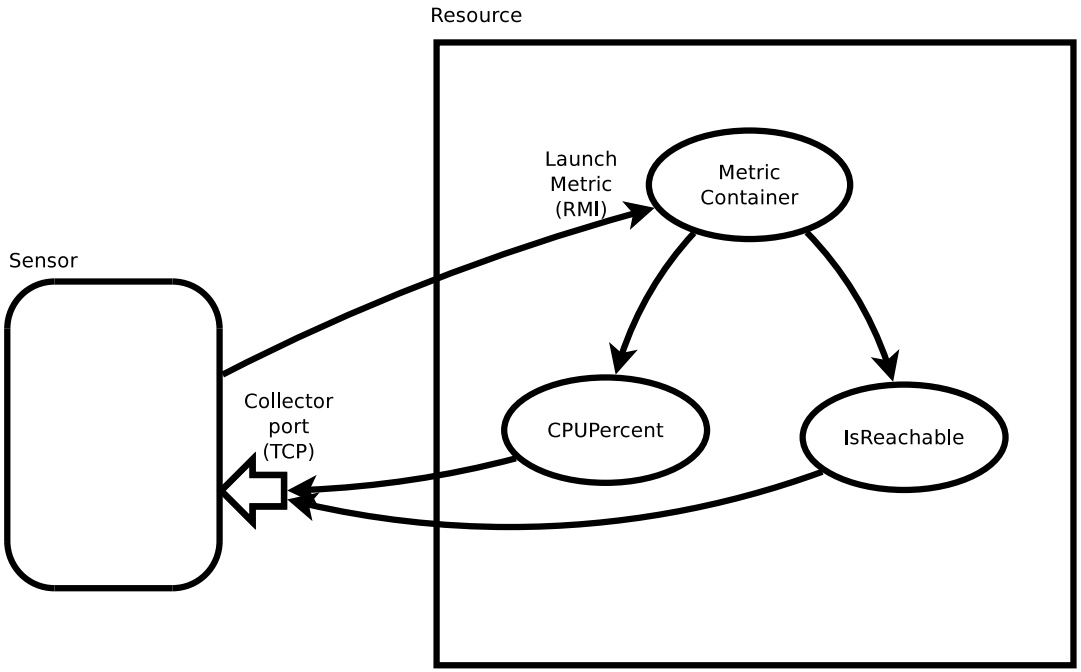
- The CollectorManager on the sensor calls a remote LaunchMetric method on the metric container
- The parameters include
  - the name of the metric mixin
  - the attributes of the mixin, encoded as a JSON object

Unfolding the Collector Endpoint



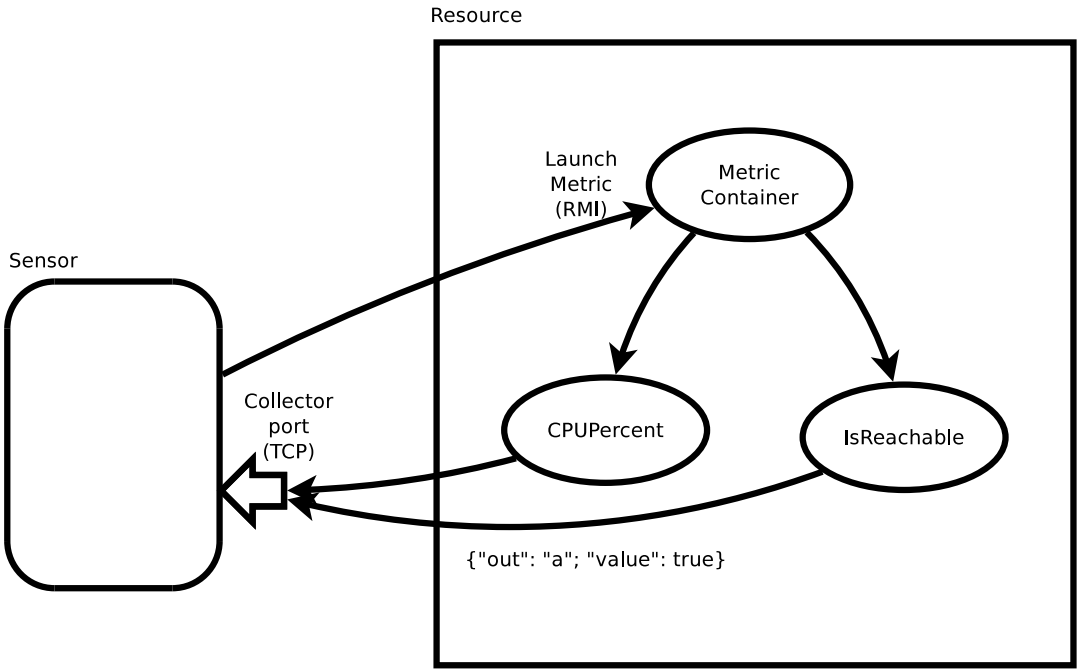
- The MetricContainer creates a metric thread (using reflection)
- The metric thread opens a connection to the Collector socket on the Sensor

Unfolding the Collector Endpoint



- All metrics associated with a Collector share the same TCP connection to the Sensor

Unfolding the Collector Endpoint

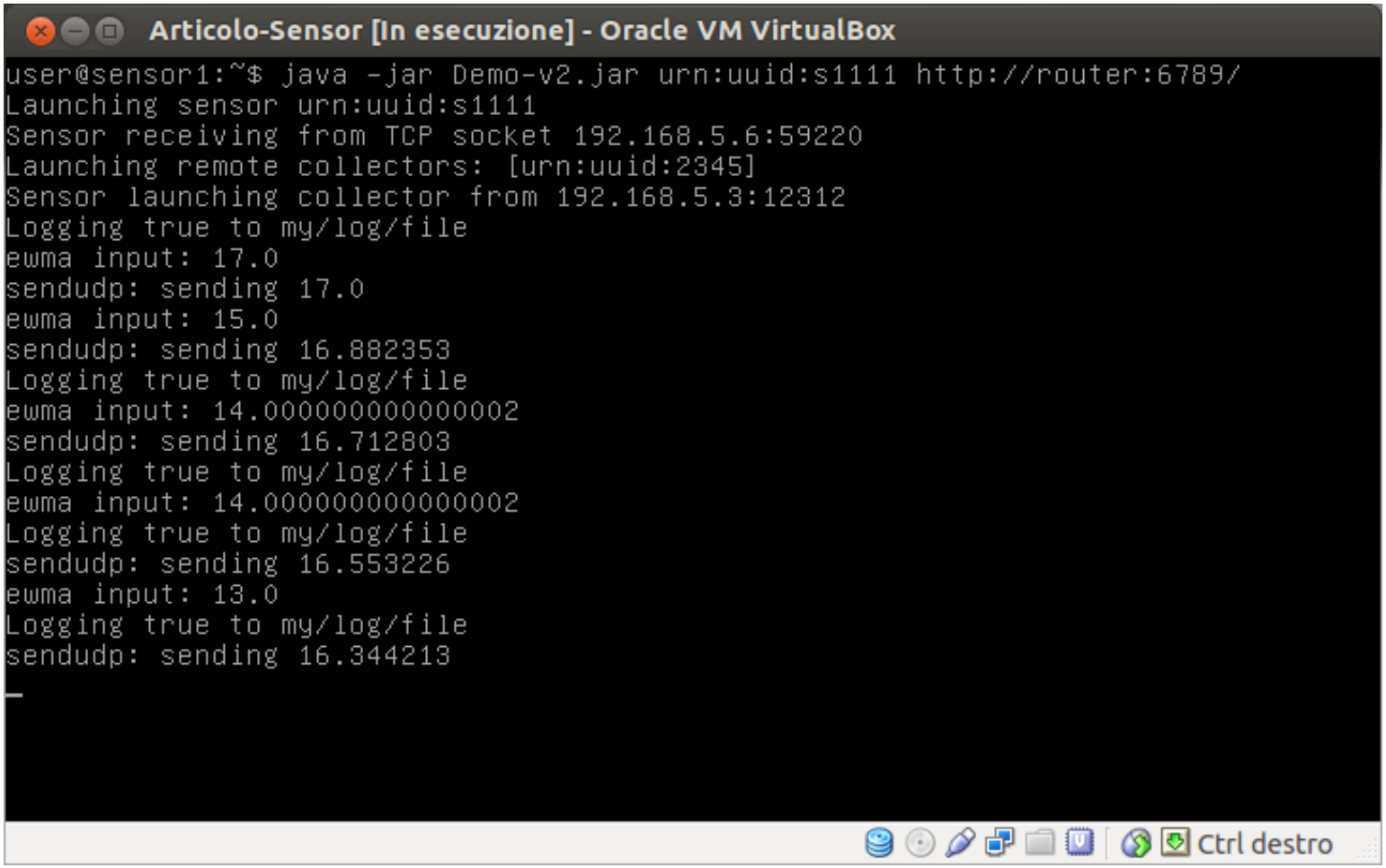


- The messages from the metric to the Sensor are JSON documents, tagged with the destination hook

Finally, the experiment

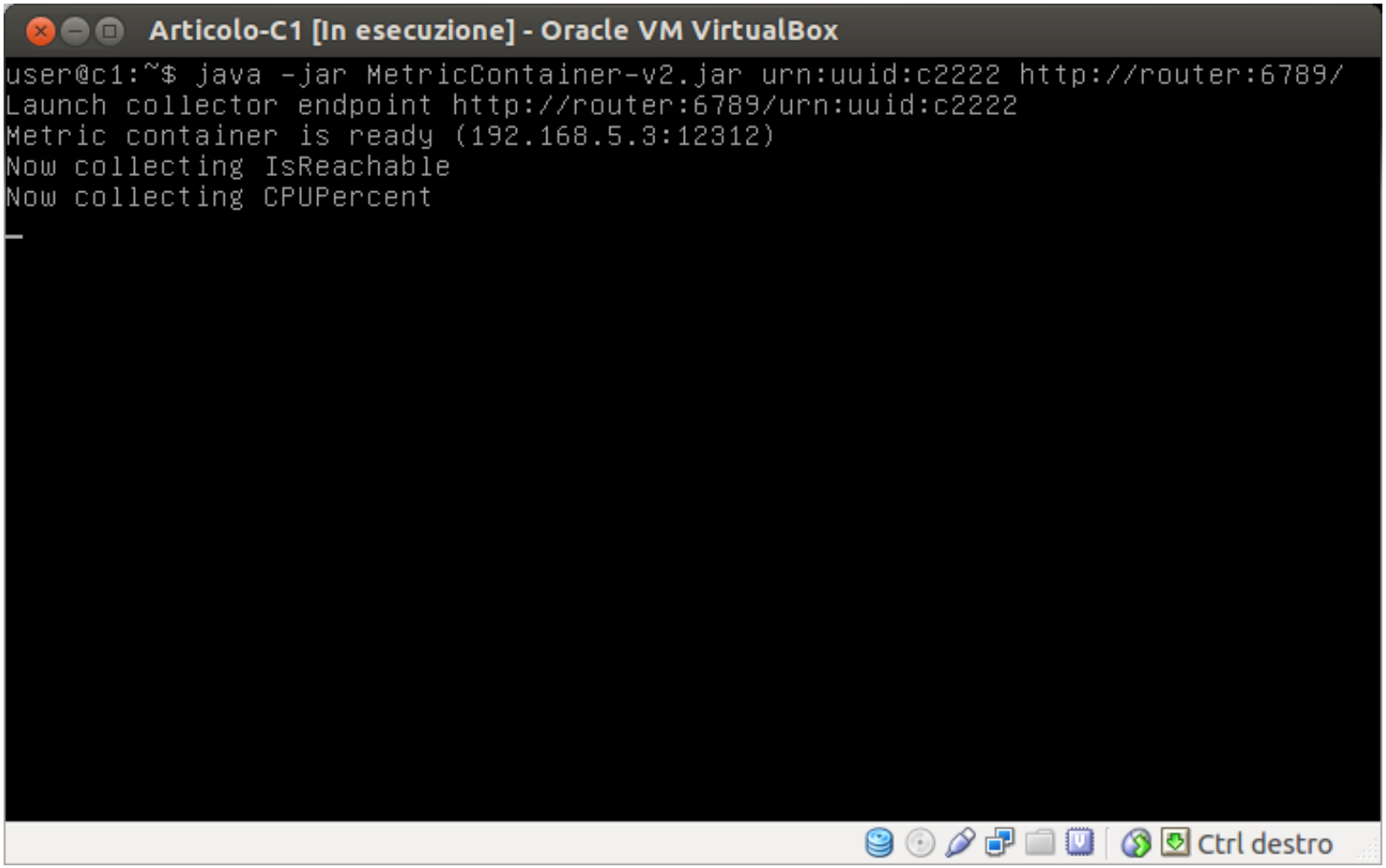
- A virtual network with guest + 3VM
- One VM acts as “coordinator”, in fact simply holding the OCCI specs as application/occi+json documents in a web server
- One VM acts as monitored resource
- One VM acts as sensor container
- The guest receives measurements through UDP

On the sensor



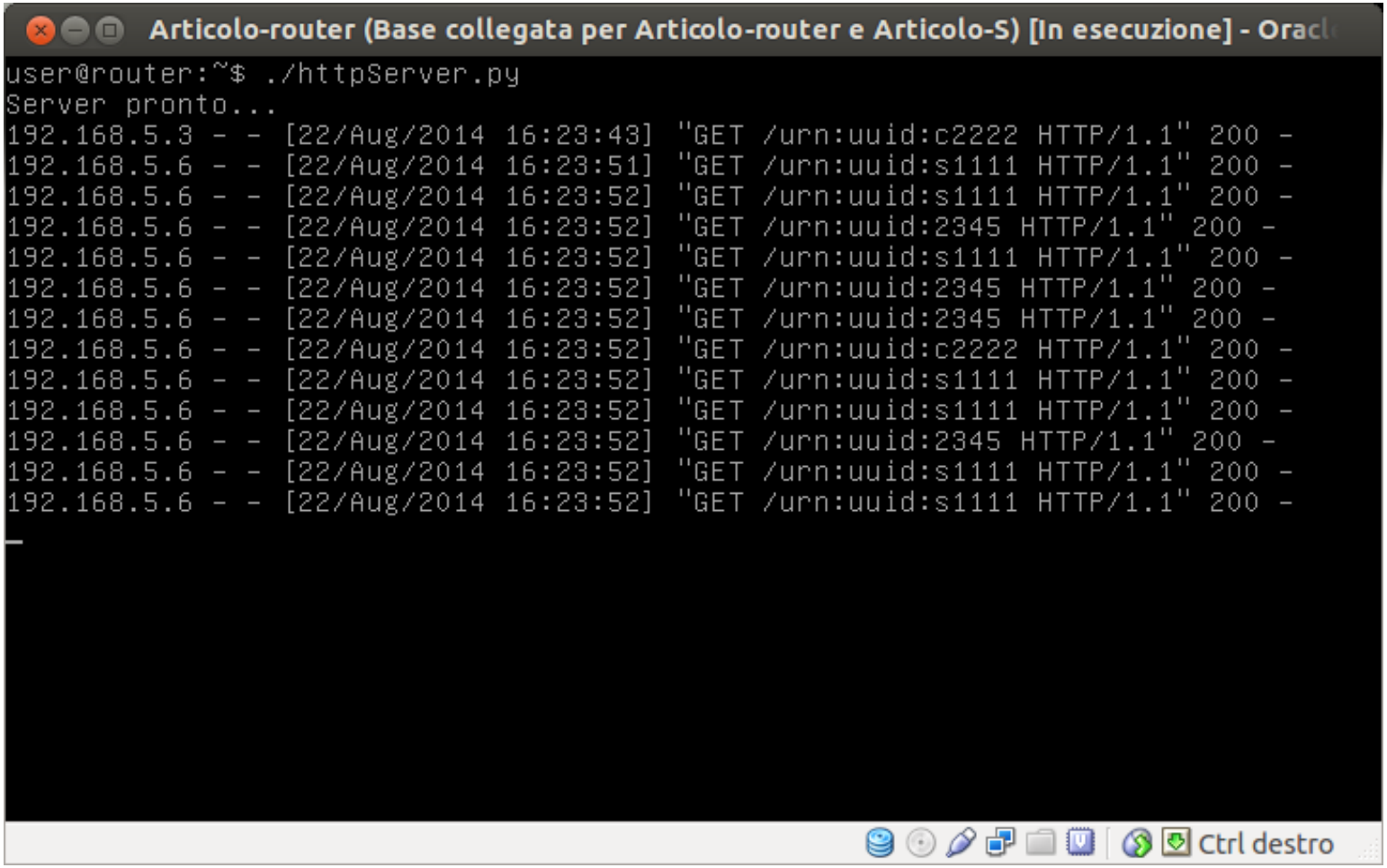
Launched by an external manager that allocates sensors to dedicated hosts

On the monitored compute resource



Launched as a consequence of the presence of a MetricContainer mixin

On the web server



- the CollectorEndpoint starts first
- the Sensor downloads the documents that describe its scope (caching!)

Conclusions of the experiment

- The proof of concept still needs some work to be finished (as a proof of concept)
- The part of the specifications concerning “named” attributes (now **hooks**) has been validated
- The Java implementation can be useful as a blueprint for a real implementation

Overall conclusions (document update)

- The MetricContainer added as a SHOULD (recommended, not strictly needed)
- Change terminology for the “named attribute” (into hook, or channel)

Overall conclusions (document update)

- The MetricContainer added as a SHOULD (recommended, not strictly needed)
- Change terminology for the “named attribute” (into hook, or channel)