Cloud Forward 2015

Automated deployment of a microservice-based monitoring infrastructure

Augusto Ciuffoletti

6 ottobre 2015

Introducing two topics

- The title:
  Automated deployment of a microservice-based monitoring infrastructure
- Microservices
- Monitoring on demand

Let us see how the two stories merge...

Introducing two topics

- The title:
  Automated deployment of a microservice-based monitoring infrastructure
- Microservices
- Monitoring on demand

Let us see how the two stories merge...

Introducing two topics

- The title:
  Automated deployment of a microservice-based monitoring infrastructure
- Microservices
- Monitoring on demand

Let us see how the two stories merge...

Introducing two topics

- The title:
  Automated deployment of a microservice-based monitoring infrastructure
- Microservices
- Monitoring on demand

Let us see how the two stories merge...

Number one: Microservices

- A design paradigm for distributed system
  - a book in O'Reilly "animal series" by S Newman
- Principles:
  - each component in the system is designed to provide one small, well defined service
  - each component is a stand alone entity that interacts with others across a network with a well defined interface

Number one: Microservices

- A design paradigm for distributed system
  - a book in O'Reilly "animal series" by S Newman
- Principles:
  - each component in the system is designed to provide one small, well defined service
  - each component is a stand alone entity that interacts with others across a network with a well defined interface

Number one: Microservices

- A design paradigm for distributed system
  - a book in O'Reilly "animal series" by S Newman
- Principles:
  - each component in the system is designed to provide one small, well defined service
  - each component is a stand alone entity that interacts with others across a network with a well defined interface

Number one: Microservices

- A design paradigm for distributed system
  - a book in O'Reilly "animal series" by S Newman
- Principles:
  - each component in the system is designed to provide one small, well defined service
  - each component is a stand alone entity that interacts with others across a network with a well defined interface

Number one: Microservices

- A design paradigm for distributed system
  - a book in O'Reilly "animal series" by S Newman
- Principles:
  - each component in the system is designed to provide one small, well defined service
  - each component is a stand alone entity that interacts with others across a network with a well defined interface

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
- simplifies development
- robustness

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
- simplifies development
- robustness

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
- robustness

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
- robustness

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
  - e.g., each component developed by a distinct team
- robustness

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
  - e.g., each component developed by a distinct team
- robustness

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
  - e.g., each component developed by a distinct team
- robustness
  - e.g., if one component fails there is a chance that the system still works

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
  - e.g., each component developed by a distinct team
- robustness
  - e.g., if one component fails there is a chance that the system still works

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
  - e.g., each component developed by a distinct team
- robustness
  - e.g., if one component fails there is a chance that the system still works

Reasons to adopt the microservices paradigm

- simplifies maintenance
  - e.g., upgrade one single component
- agility in deployment
  - e.g., to scale up or down
- each component may use a different technology
  - e.g., for technical or performance reasons
- simplifies development
  - e.g., each component developed by a distinct team
- robustness
  - e.g., if one component fails there is a chance that the system still works

Number two: Cloud monitoring

A cloud user wants to have a functional feedback from cloud sourced resources:

- not necessarily to verify service quality
  - control a scalable resource,
  - provide feedback to the users,
  - trigger compensating actions
- NIST indicates monitoring as one of the distinctive features of cloud computing

Number two: Cloud monitoring

A cloud user wants to have a functional feedback from cloud sourced resources:

- not necessarily to verify service quality
  - control a scalable resource,
  - provide feedback to the users,
  - trigger compensating actions
- NIST indicates monitoring as one of the distinctive features of cloud computing

Number two: Cloud monitoring

A cloud user wants to have a functional feedback from cloud sourced resources:

- not necessarily to verify service quality
  - control a scalable resource,
  - provide feedback to the users,
  - trigger compensating actions
- NIST indicates monitoring as one of the distinctive features of cloud computing

Number two: Cloud monitoring

A cloud user wants to have a functional feedback from cloud sourced resources:

- not necessarily to verify service quality
  - control a scalable resource,
  - provide feedback to the users,
  - trigger compensating actions
- NIST indicates monitoring as one of the distinctive features of cloud computing

Number two: Cloud monitoring

A cloud user wants to have a functional feedback from cloud sourced resources:

- not necessarily to verify service quality
  - control a scalable resource,
  - provide feedback to the users,
  - trigger compensating actions
- NIST indicates monitoring as one of the distinctive features of cloud computing

Number two: Cloud monitoring

A cloud user wants to have a functional feedback from cloud sourced resources:

- not necessarily to verify service quality
  - control a scalable resource,
  - provide feedback to the users,
  - trigger compensating actions
- NIST indicates monitoring as one of the distinctive features of cloud computing

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
- Resource agnostic

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic
  - Basic functionalities and unlimited plugable extensions

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic
  - Basic functionalities and unlimited pluggable extensions

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic
  - Basic functionalities and unlimited pluggable extensions

Our option: on demand monitoring

- Provide monitoring as part of the service
- Give the user wide possibilities to configure a monitoring infrastructure
  - Which metrics are captured and how data are preprocessed and retrieved
- Scale from simple to complex infrastructures
  - Do not overkill the problem in simple cases
  - Cope with complex infrastructures
- Resource agnostic
  - Basic functionalities and unlimited pluggable extensions

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
- The *on demand* nature requires agility in deployment

There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
- The *on demand* nature requires agility in deployment

There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment

There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment

There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment
  - the cloud user that obtains a new resource may want to monitor it

There is a match between microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment
  - the cloud user that obtains a new resource may want to monitor it

There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment
  - the cloud user that obtains a new resource may want to monitor it

There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment
  - the cloud user that obtains a new resource may want to monitor it

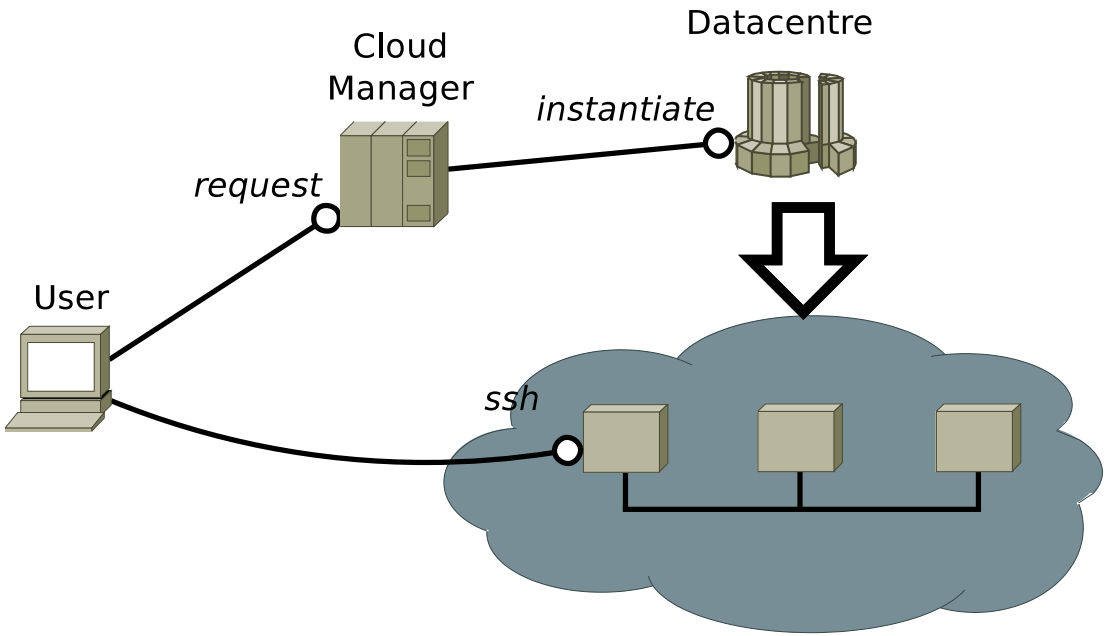
There is a match between
microservices and on demand monitoring

Find a match

- Monitoring is by nature split into small components (remember Nagios)
  - monitoring probes are small components, possibly embedded
  - monitoring data crosses a pipe of processors (anonymization, aggregation etc)
  - data is finally published using an endpoint reachable from the outside (database, web service)
- Each component is supported by a specific technology
  - e.g., network monitoring vs storage monitoring
- The *on demand* nature requires agility in deployment
  - the cloud user that obtains a new resource may want to monitor it

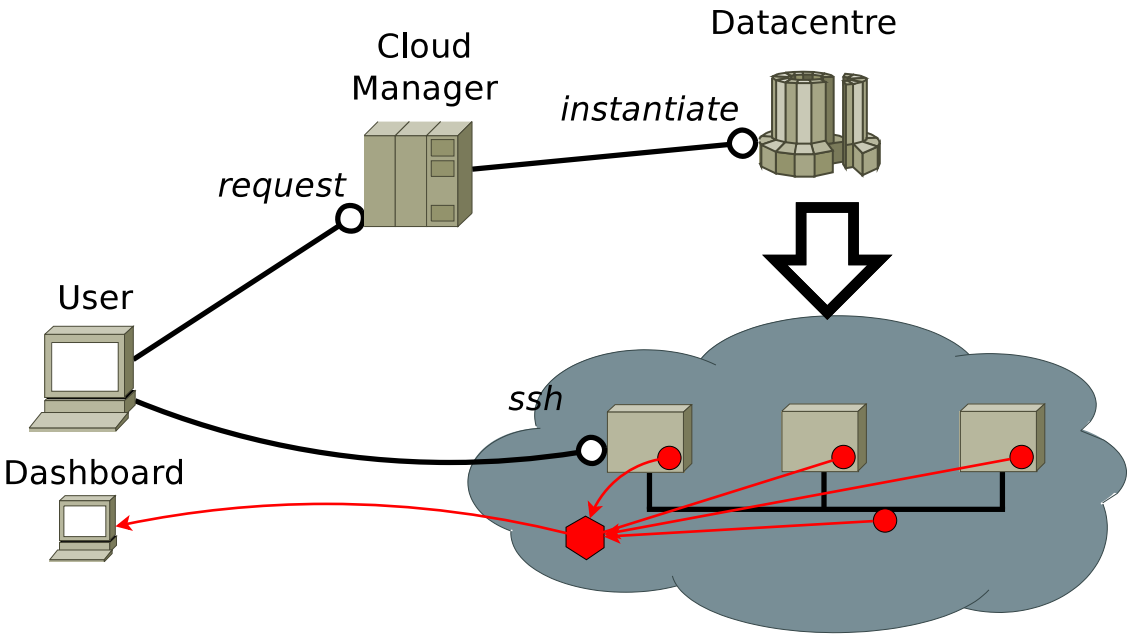
There is a match between
microservices and on demand monitoring

Summarizing: a IaaS provisioning



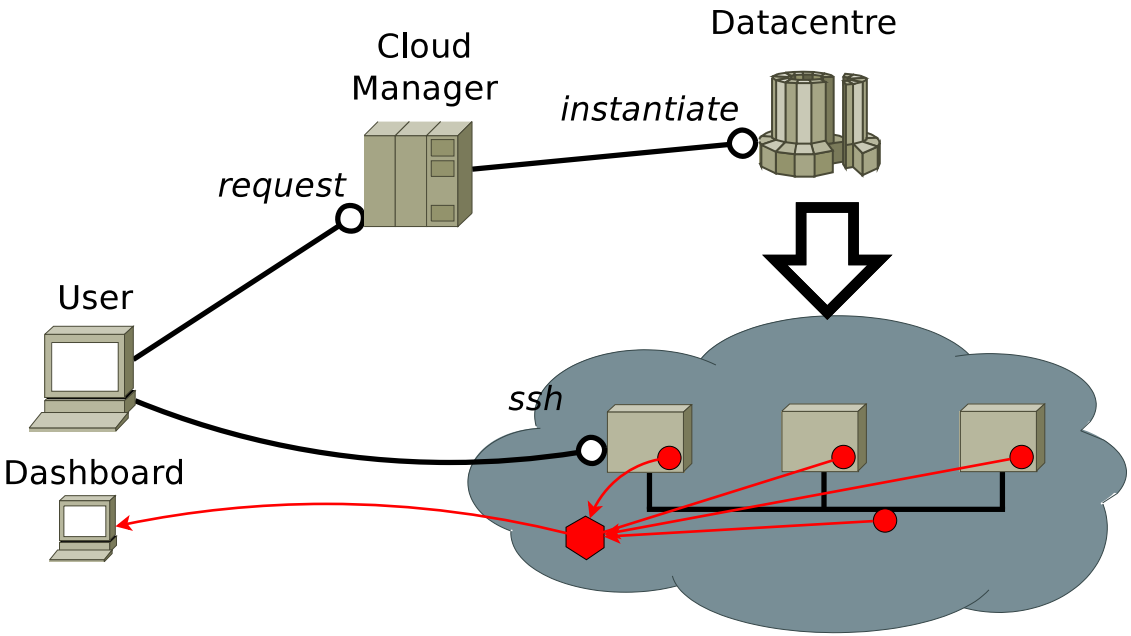
- IaaS is just for the example: could be PaaS or anything else

A monitoring infrastructure



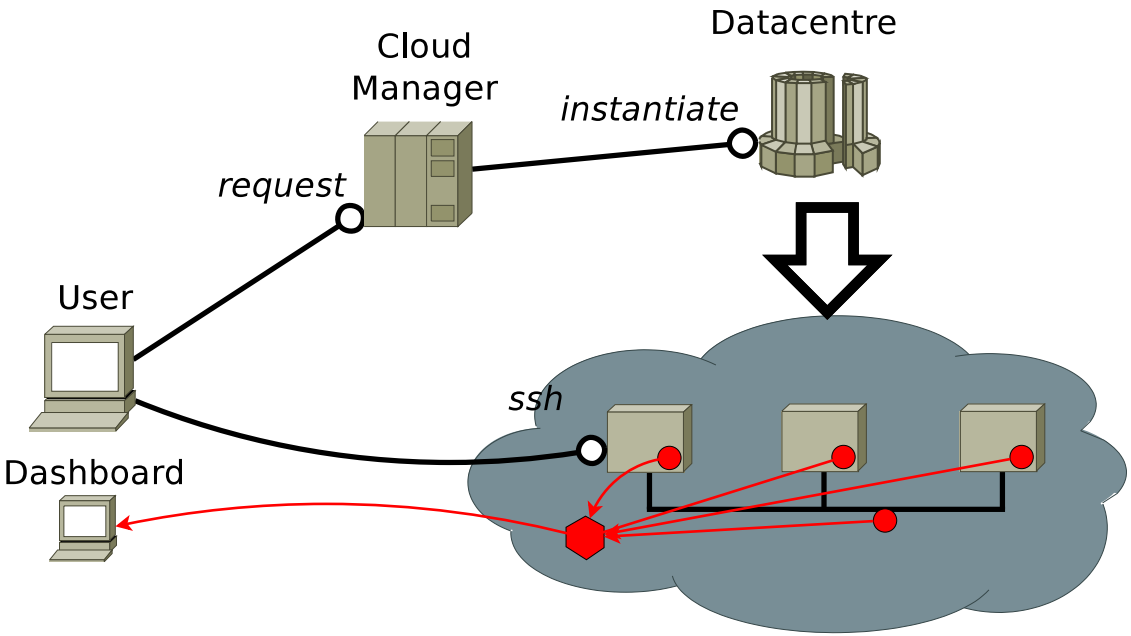
- Adding a monitoring infrastructure:
  - probes that collect monitoring data (collectors)
  - a device that processes monitoring data (sensor)

A monitoring infrastructure



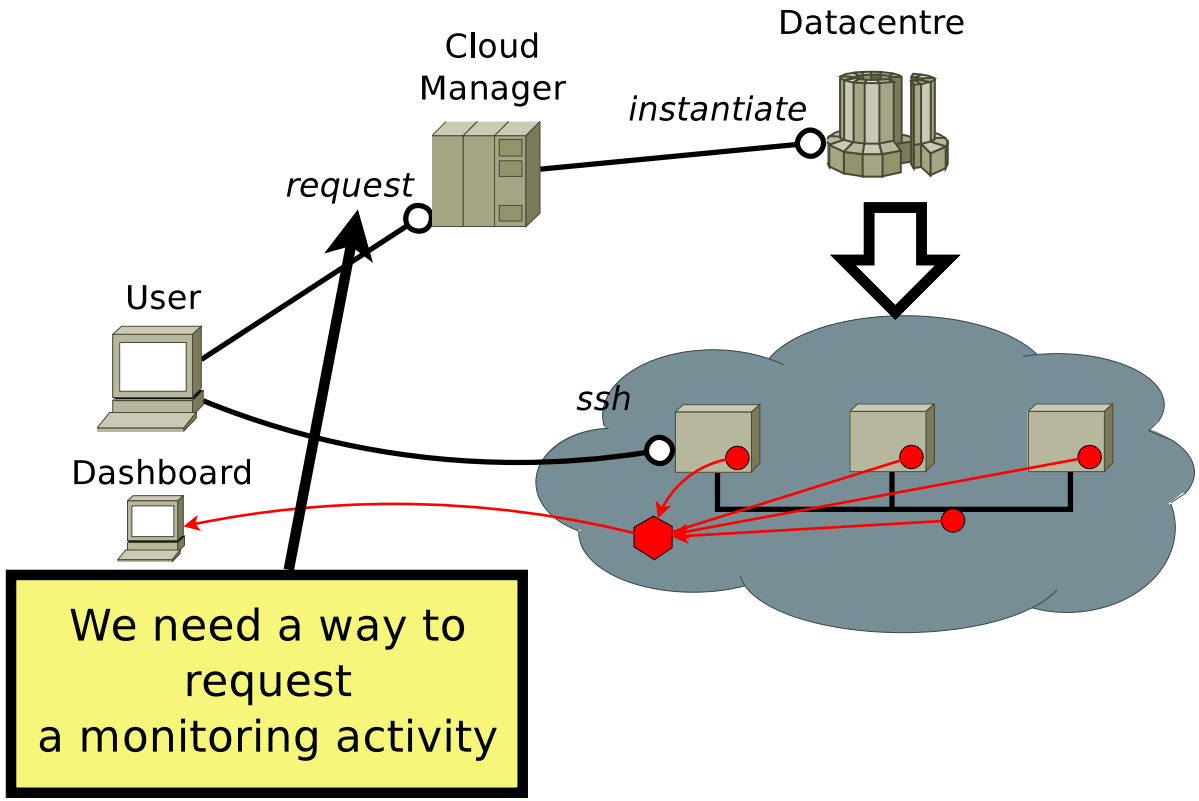
- Adding a monitoring infrastructure:
  - probes that collect monitoring data (collectors)
  - a device that processes monitoring data (sensor)

A monitoring infrastructure



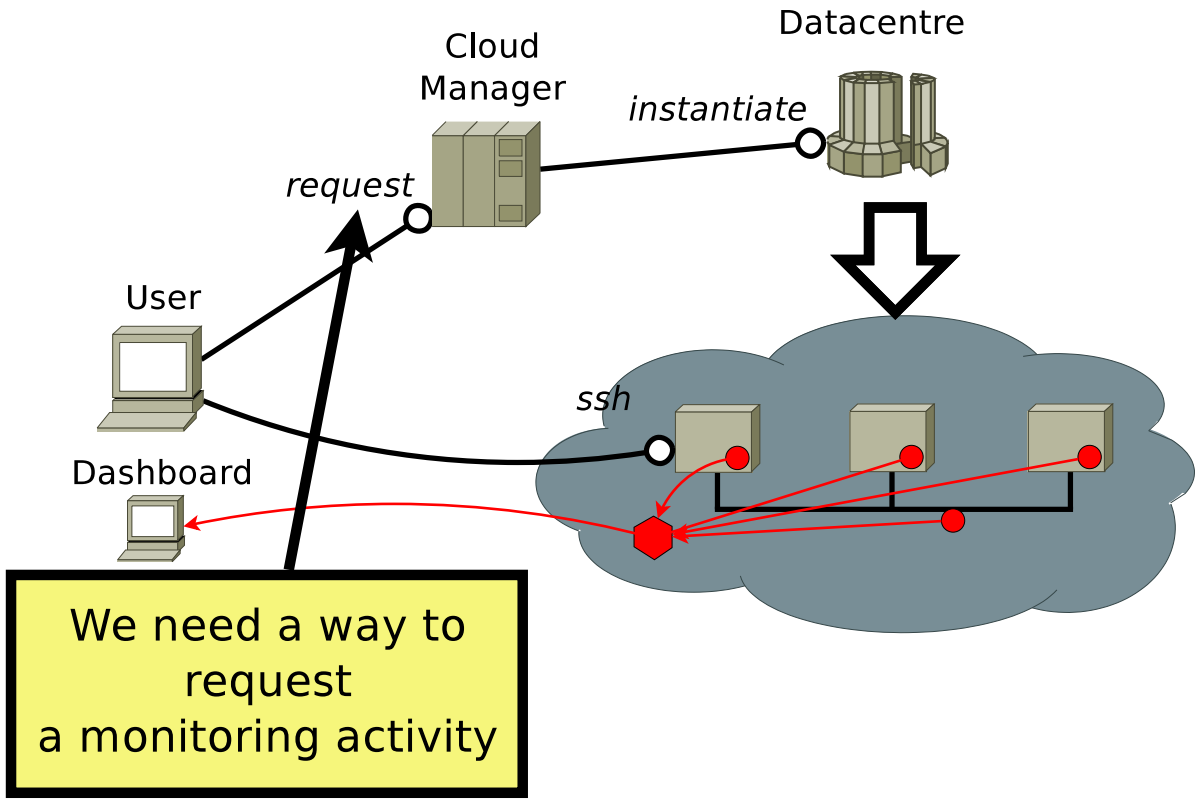
- Adding a monitoring infrastructure:
  - probes that collect monitoring data (collectors)
  - a device that processes monitoring data (sensor)

A cloud interface



- we need to design an interface
- an open standard exists: OCCI

A cloud interface



- we need to design an interface
- an open standard exists: OCCI

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances

- The interface is extensible:

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances

- The interface is extensible:

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - Attributes, whose values define instances
  - Actions that model dynamic change
- The interface is extensible:

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:
  - a type can be extended thus adding new attributes to the standard ones
  - an instance can be modified using new attributes

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:
  - a type can be subtyped, thus adding new attributes to the standard ones
  - an instance can be modified using mixins

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:
  - a type can be subtyped, thus adding new attributes to the standard ones
  - an instance can be modified using mixins

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:
  - a type can be subtyped, thus adding new attributes to the standard ones
  - an instance can be modified using mixins

Open Cloud Computing Interface basics

- The interface is REST, therefore web-oriented
- The items accessed across the interface are entities
- One type of entity is the resource
- Another is the link, that connects two resources
- A type is characterized by standard features of the instances
  - attributes whose values define instances
  - actions that model dynamic change
- The interface is extensible:
  - a type can be subtyped, thus adding new attributes to the standard ones
  - an instance can be modified using mixins

An OCCI model for monitoring

- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity
- Legenda:

An OCCI model for monitoring

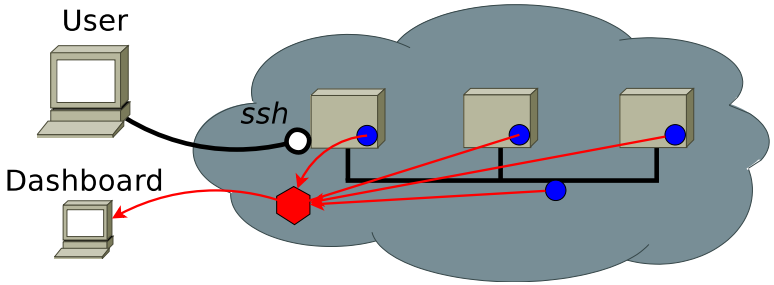
- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity
- Legenda:

An OCCI model for monitoring

- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity
- Legenda:
  - the sensors (red) in an OCCI resource
  - the collectors (blue) are OCCI links
  - computing boxes and the network are OCCI resources too

An OCCI model for monitoring

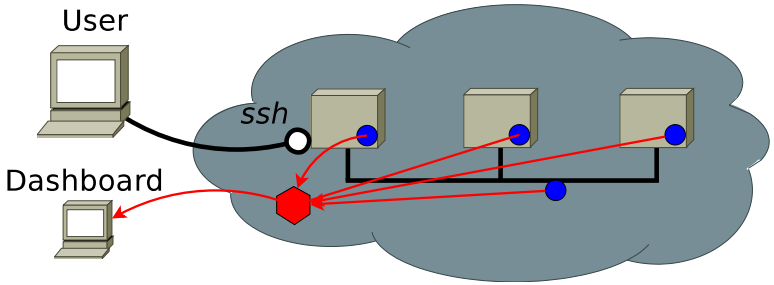
- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity



- Legenda:
  - the sensor (red) is an OCCI resource
  - the collectors (blue) are OCCI links
  - computing boxes and the network are OCCI resources too

An OCCI model for monitoring

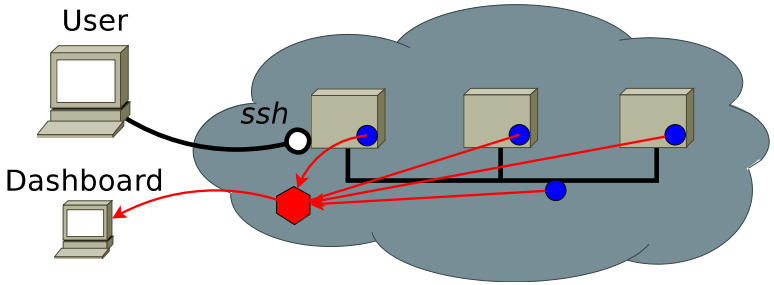
- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity



- Legenda:
  - the sensor (red) is an OCCI resource
  - the collectors (blue) are OCCI links
  - computing boxes and the network are OCCI resources too

An OCCI model for monitoring

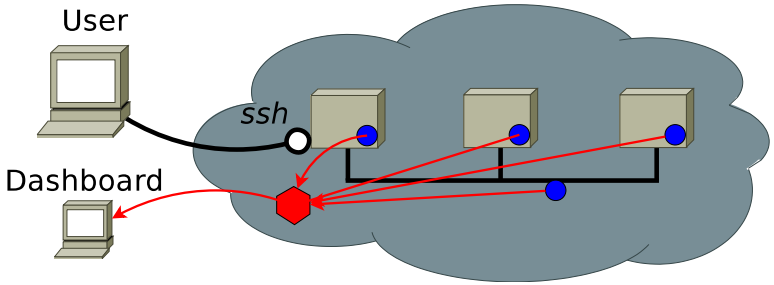
- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity



- Legenda:
  - the sensor (red) is an OCCI resource
  - the collectors (blue) are OCCI links
  - computing boxes and the network are OCCI resources too

An OCCI model for monitoring

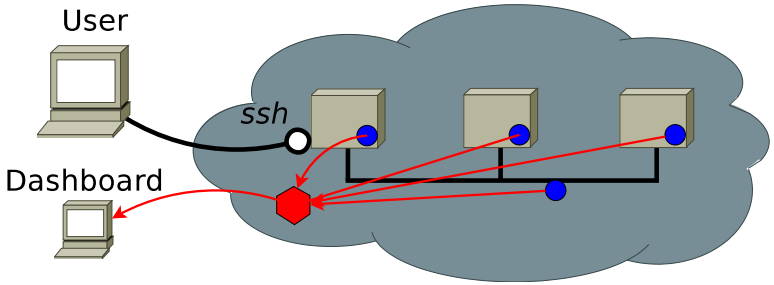
- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity



- Legenda:
  - the sensor (red) is an OCCI resource
  - the collectors (blue) are OCCI links
  - computing boxes and the network are OCCI resources too

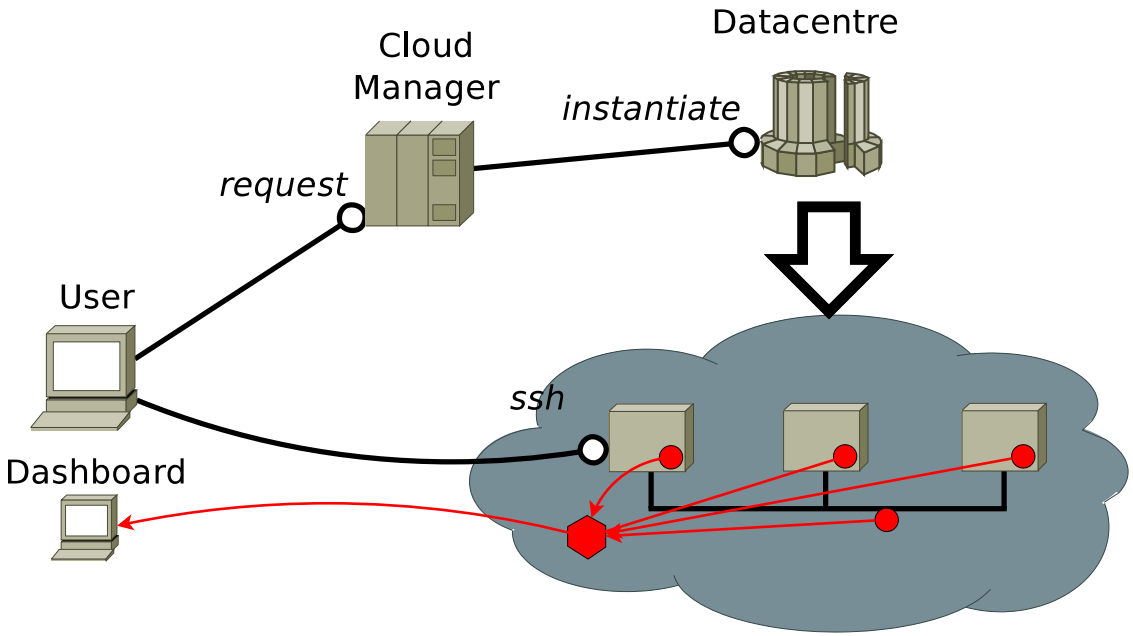
An OCCI model for monitoring

- A Sensor is a subtype of the Resource type
- A Collector is a subtype of the Link type
- Add Mixins to specify the type of activity



- Legenda:
  - the sensor (red) is an OCCI resource
  - the collectors (blue) are OCCI links
  - computing boxes and the network are OCCI resources too

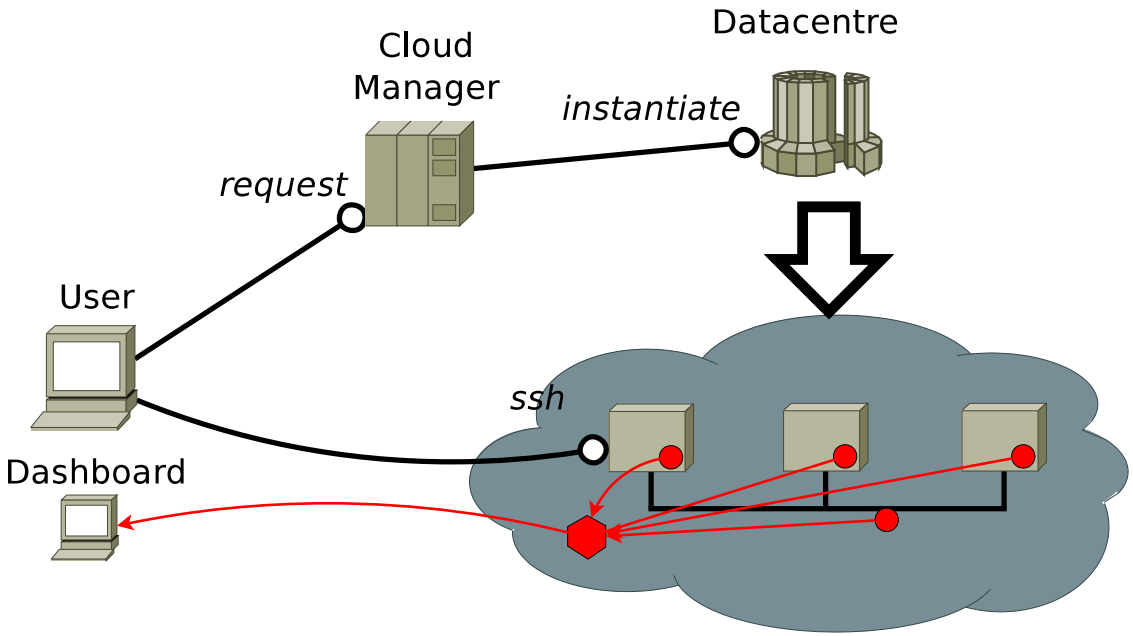
How do we do that?



- We want to study the big arrow in the figure

How do we implement a monitoring infrastructure starting form OCCI entities

How do we do that?



- We want to study the big arrow in the figure

How do we implement a monitoring infrastructure starting form OCCI entities

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker
  - demo is easily reproduceable/shareable
  - efficient to run on a single laptop
  - upgradeable to a real deployment

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker
  - demo is easely reproduceable/shareable
  - efficient to run on a single laptop
  - upgradeable to a real deployment

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker
  - demo is easely reproduceable/shareable
  - efficient to run on a single laptop
  - upgradeable to a real deployment

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker
  - demo is easely reproduceable/shareable
  - efficient to run on a single laptop
  - upgradeable to a real deployment

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker
  - demo is easely reproduceable/shareable
  - efficient to run on a single laptop
  - upgradeable to a real deployment

Vintage programming but...

- For an early prototype we selected seasoned technologies: Java/Unix sockets/Unix Pipes
  - not bound to specific programming tools
  - better solutions do exist
- As a virtualization platform we selected Docker
  - demo is easely reproduceable/shareable
  - efficient to run on a single laptop
  - upgradeable to a real deployment

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view
  - One *compute* resource
  - One *sensor* resource
  - One *collector link*

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view
  - One *compute resource*
  - One *sensor resource*
  - One *collector link*

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view
  - One *compute resource*
  - One *sensor resource*
  - One *collector link*

Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view
  - One *compute resource*
  - One *sensor resource*
  - One *collector link*

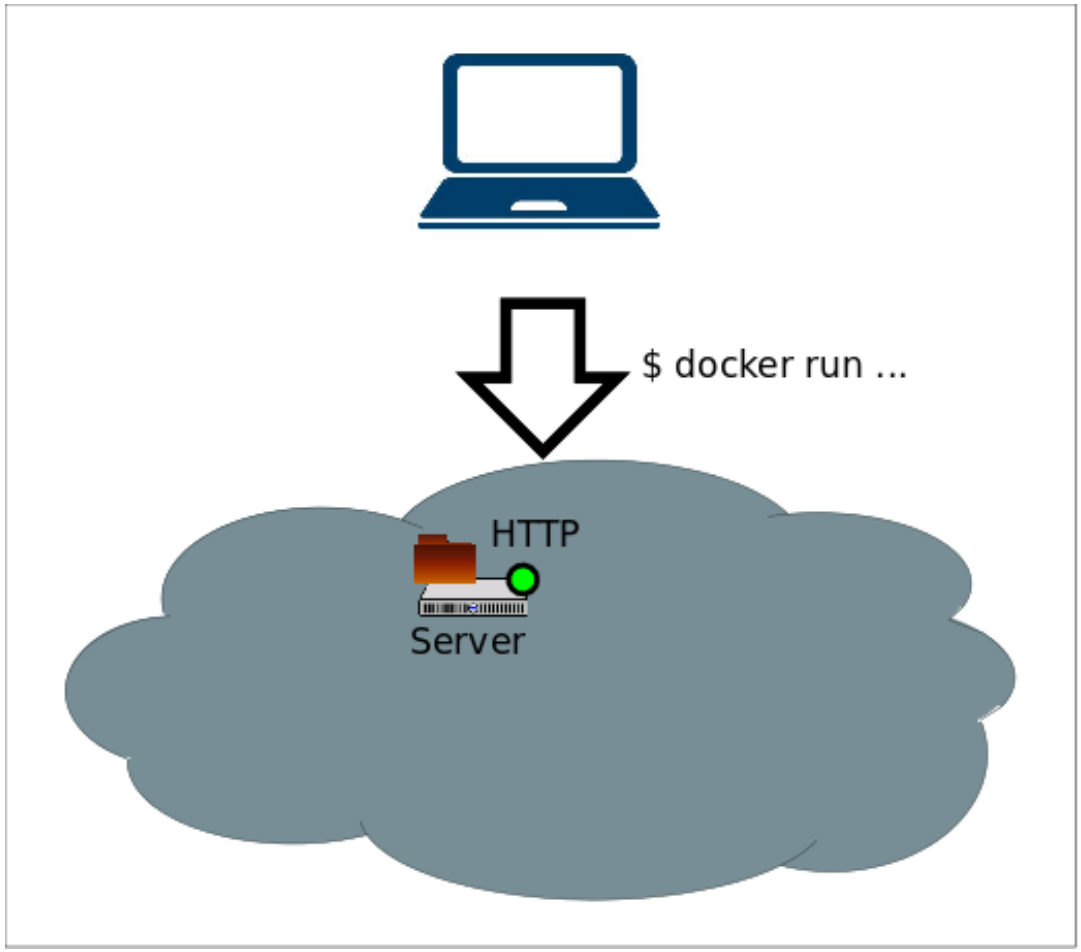
Deploying a demo infrastructure

- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view
  - One *compute resource*
  - One *sensor resource*
  - One *collector link*

Deploying a demo infrastructure

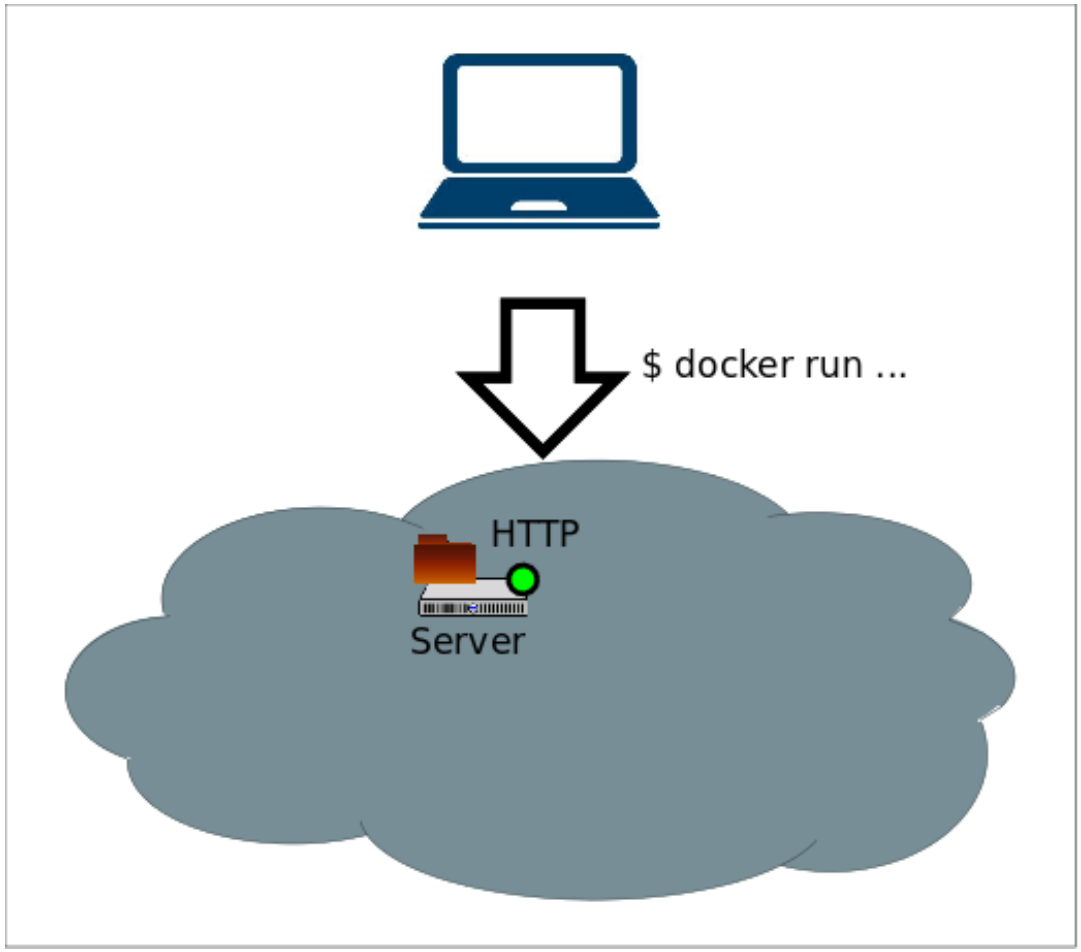
- The minimal architecture (4 dockers):
  - one HTTP server to host the OCCI entities descriptions
  - one dashboard to display monitoring data
  - one monitored compute resource
  - one sensor resource
- In OCCI view
  - One *compute resource*
  - One *sensor resource*
  - One *collector link*

The web server



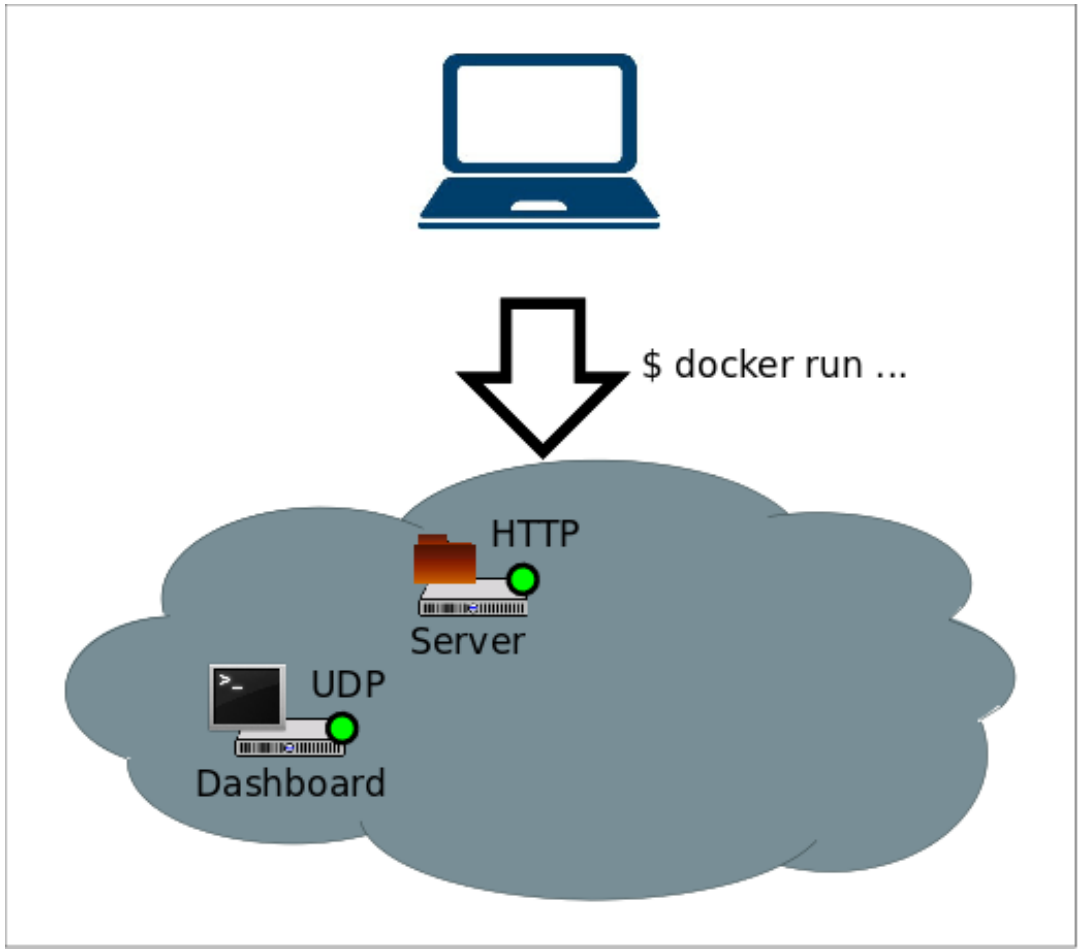
- The web server is deployed (with OCCI docs)
- this is part of the cloud infrastructure

The web server



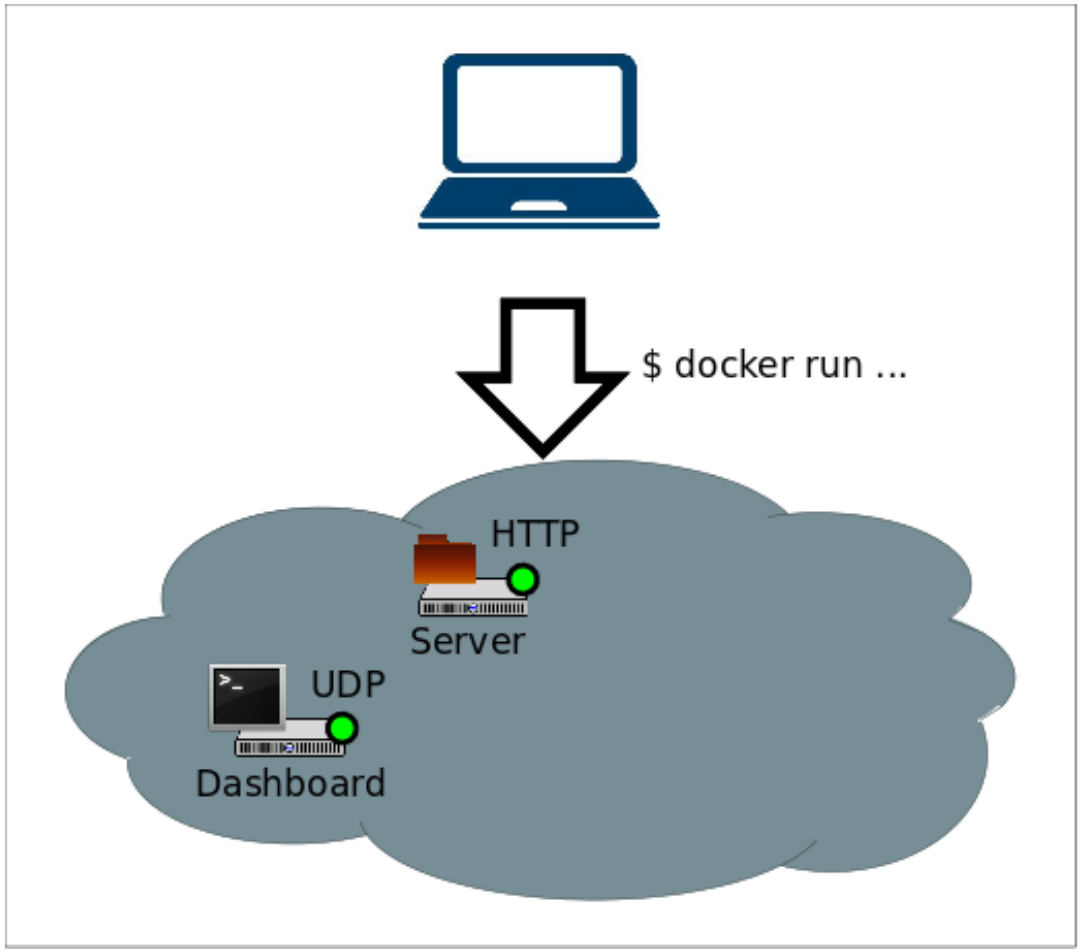
- The web server is deployed (with OCCI docs)
- this is part of the cloud infrastructure

The dashboard



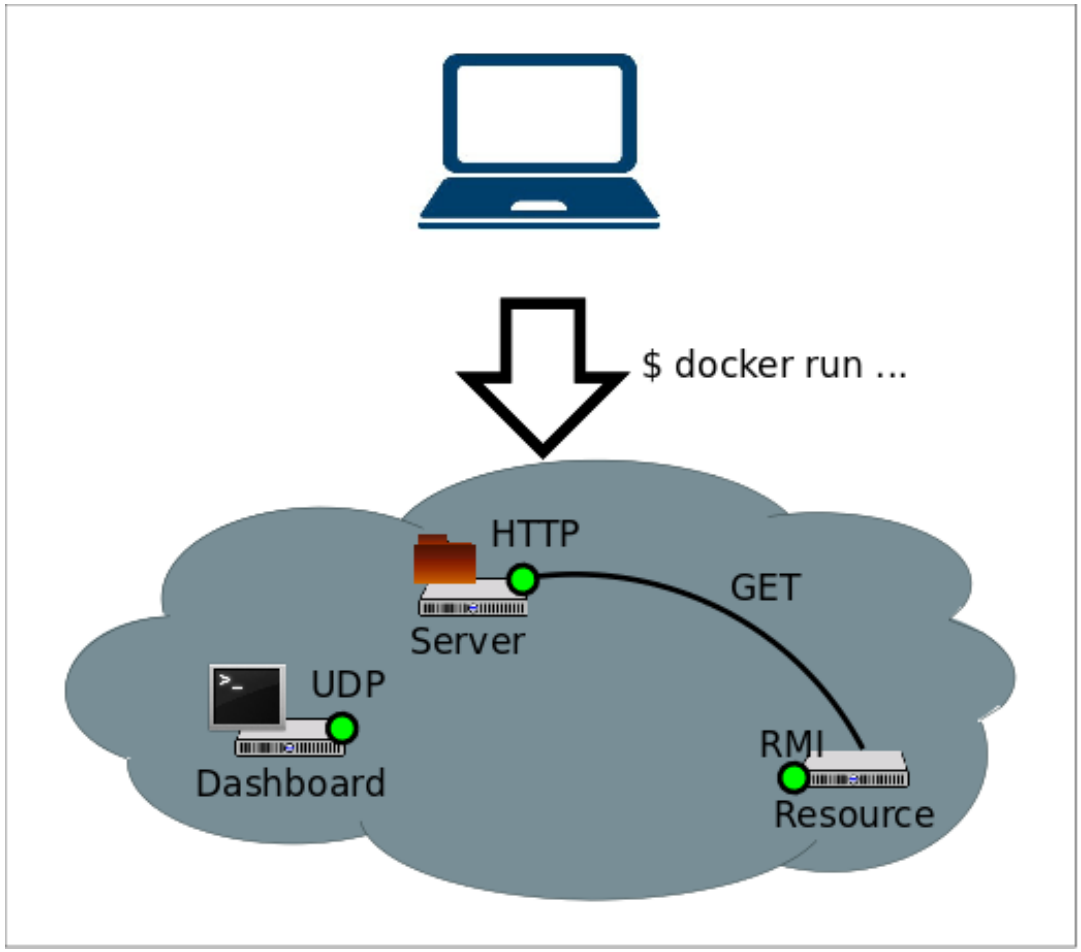
- The dashboard is deployed (UDP to receive data)
- this is on user premises

The dashboard



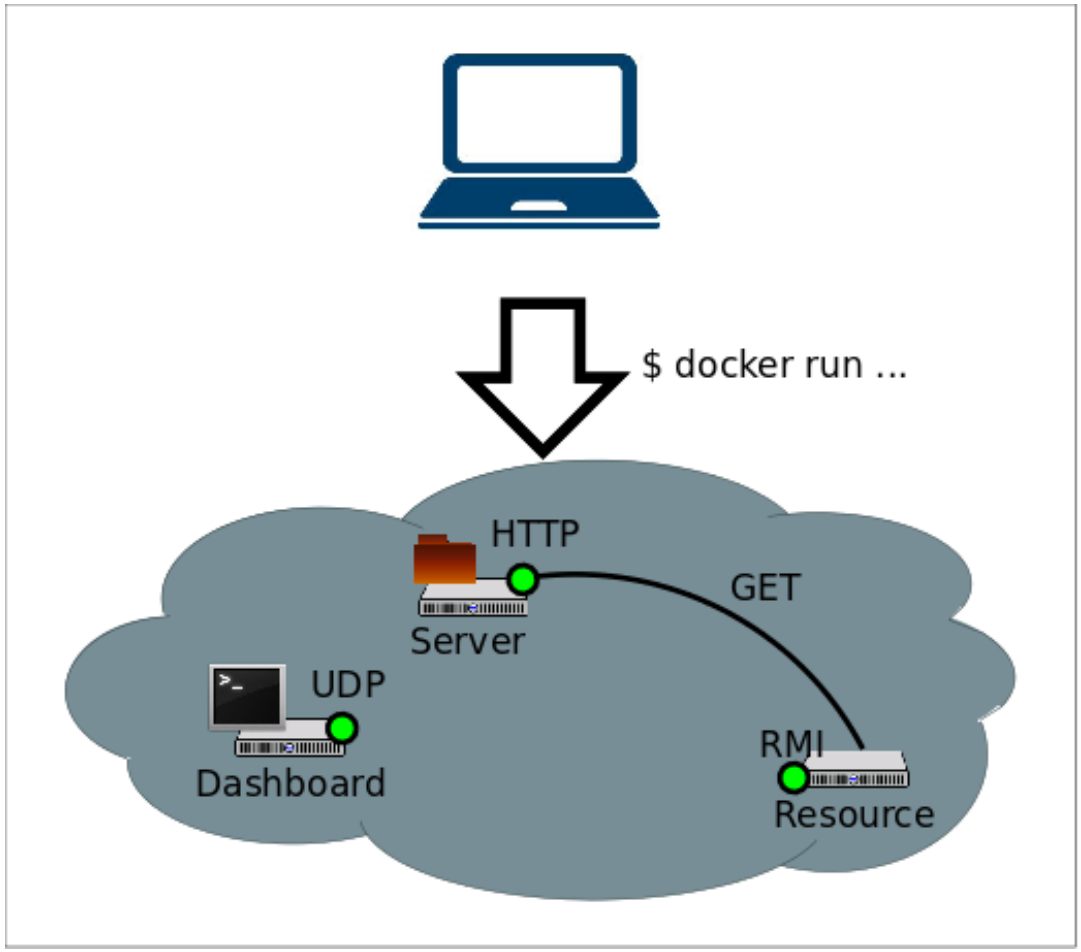
- The dashboard is deployed (UDP to receive data)
- this is on user premises

The computing resource



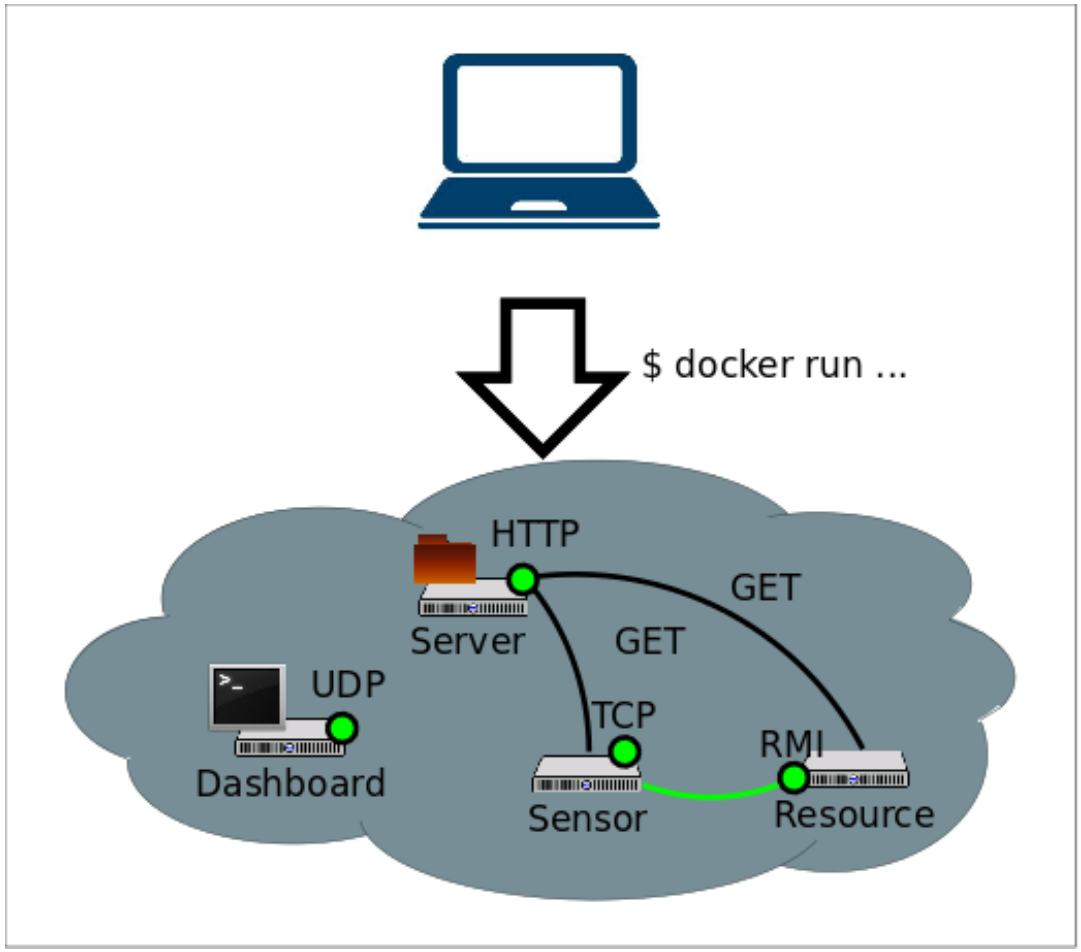
- The virtual computing resource is deployed
- the RMI interface is for configuration

The computing resource



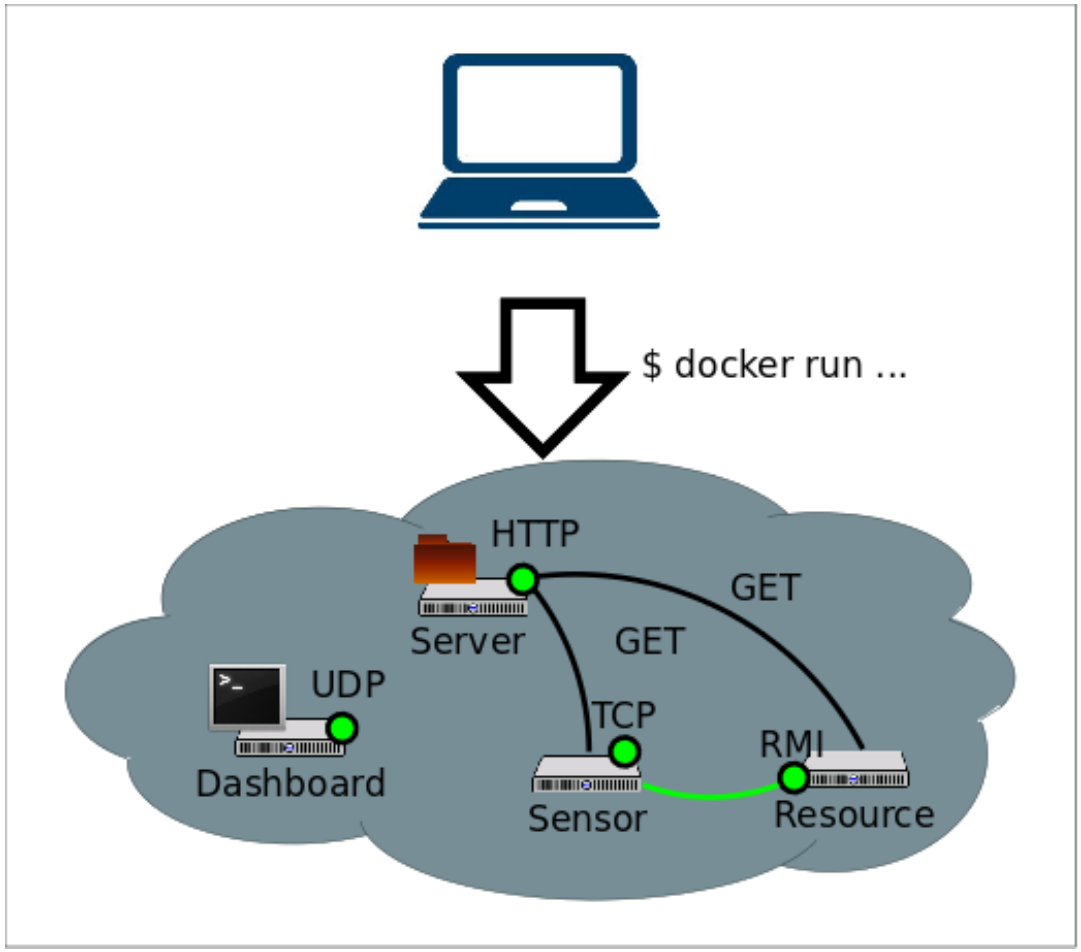
- The virtual computing resource is deployed
- the RMI interface is for configuration

The sensor



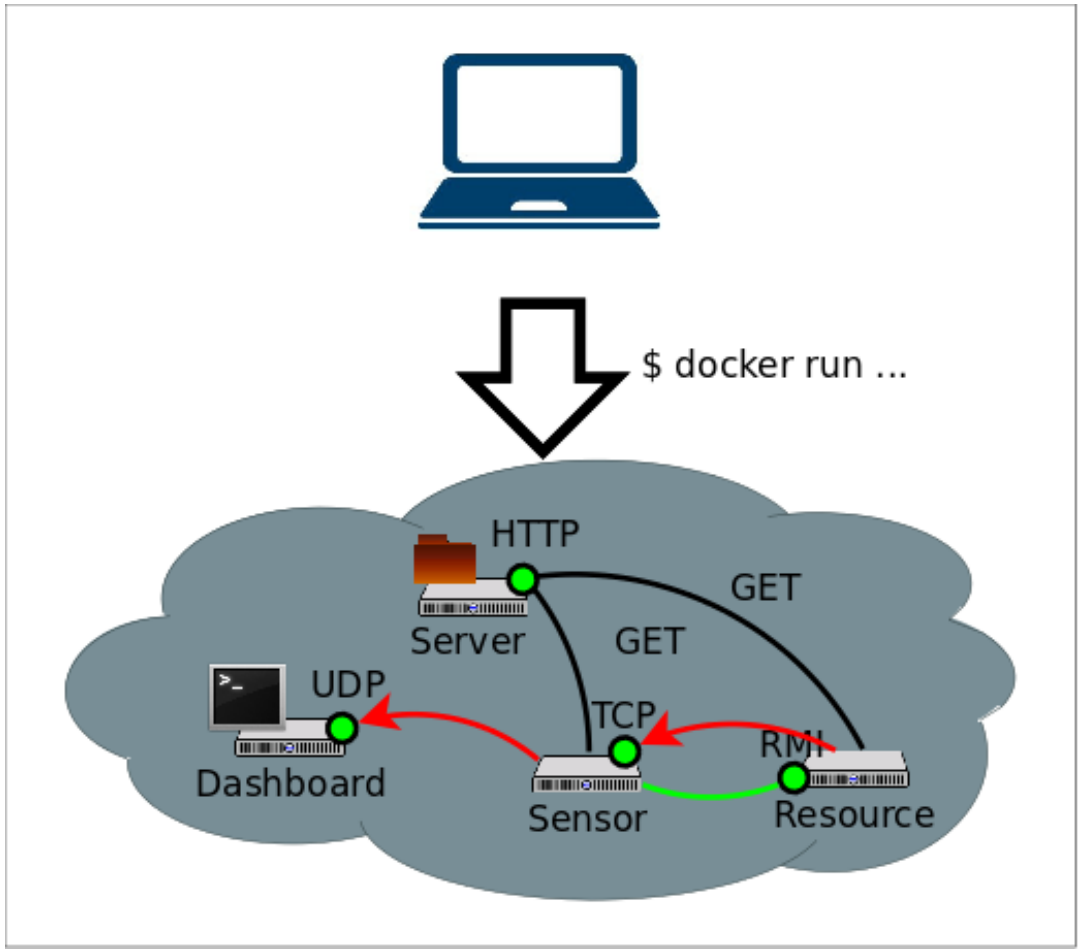
- It is the resource that implements the monitoring
- uploads the collector across the RMI interface

The sensor



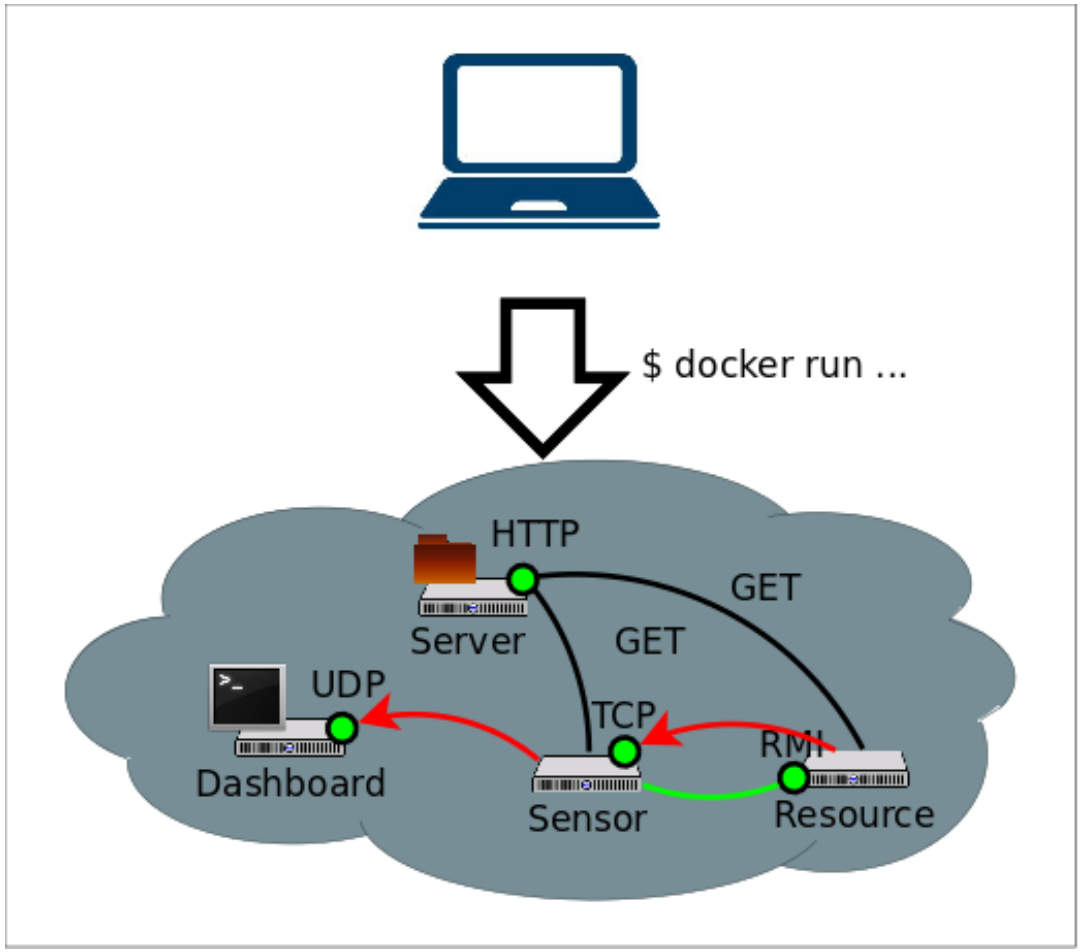
- It is the resource that implements the monitoring
- uploads the collector across the RMI interface

GO!



- The dashboard receives the probe measurements...
- ...filtered by the sensor

GO!



- The dashboard receives the probe measurements...
- ...filtered by the sensor

Conclusions

- Microservices are appropriate for distributed monitoring
- The OCCI API is appropriate as the interface
- A prototype is available on dockerhub (look for **occimon-live**)
- BYOD demo on stage tonight at 17:45

Conclusions

- Microservices are appropriate for distributed monitoring
- The OCCI API is appropriate as the interface
- A prototype is available on dockerhub (look for **occimon-live**)
- BYOD demo on stage tonight at 17:45

Conclusions

- Microservices are appropriate for distributed monitoring
- The OCCI API is appropriate as the interface
- A prototype is available on dockerhub (look for **occimon-live**)
- BYOD demo on stage tonight at 17:45

Conclusions

- Microservices are appropriate for distributed monitoring
- The OCCI API is appropriate as the interface
- A prototype is available on dockerhub (look for **occimon-live**)
- BYOD demo on stage tonight at 17:45