avassa

Observability in distributed edge: The full story

Imagine you manage thousands of edge sites that run business-critical applications. These could be in-store applications, property management solutions, software within medical equipment, or industrial IoT apps.

Regardless, all edge applications run physically near the location where they fulfill their purpose. Because of this, it is vital to have continuous proactive insight into their performance and to be able to restore any degradations as fast as possible. This level of monitoring is not new in the IT industry. We have always needed monitoring systems within IT, but historically some have been more successful than others. We can't simply throw our existing IT monitoring solutions at the edge and hope they work.

Table of Contents

Edgy observability? 1

Where are we on monitoring and observability today? 2

The naive approach 2
The scientific approach 3
The four observability pillars 5
- Bottom-up pillars 5
- Top-down pillars 6
- Bridging pillars 6
- Action pillars 7

What is different with edge observability? 9

Six solution principles for edge observability 11

Principle 1: Application centricity 11
Principle 2: Edge site awareness 11
Principle 3: Deploy and operations perspectives 13
Principle 4: Infrastructure and application views 14
Principle 5: All four pillars combined 15
Principle 6: Managing scale 17

Conclusion 19

01 Edgy observability?

Before moving on, let us have a look at what we mean by “observability” and how it compares with “monitoring”.

What is the ultimate goal for observability?

> **Observability is the ability to answer any question about your application, at any time, no matter how complex your infrastructure is.**

Is that not just a more fancy term for monitoring then?

Monitoring tells you that something is wrong, not why. The classical monitoring system works with what you already know is problematic, your “known knowns.” Observability adds insights into what happened, the ability to understand the underlying reasons. Furthermore, classical monitoring systems often leave out the application’s user experience, such as synthetic monitoring of response times.

Observability should help you focus on what matters, not digging around amongst metrics.

It is an inherent capability to inspect and understand systems. It moves beyond looking at the individual component and looks at the outcome of the system as a whole. With an observability mindset, you pay attention to the overall system and the user’s experience, not each component of it.

In this white paper, we will look at four things

1. Give some reflections on the **current state** of observability and monitoring in general
2. Define **four pillars** for a sound observability solution
3. Summarize the characteristics and **challenges for edge observability**
4. Define what is needed for **successfully assuring application health at the edge**

02 Where are we on monitoring and observability today?

The naive approach

Regardless of whether you have labelled your solution as “monitoring” or “observability,” what truly matters is the value you get out of it.

And simply renaming a solution as one that delivers observability does not mean it will, or that it will generate any value. Unfortunately, that sort of shortcut is all too common these days.

To get actual value you need quality monitoring data, instead of focusing on how it is transported on the wire or stored.

There have been too many discussions describing monitoring solutions as a bucket in which you collect as many events as possible, and too much focus on telemetry protocols and the data analytics platform. Many choose to stay blind to reality, hoping that the analytic layer will give them free lunches. Often, when this approach is taken, the corresponding dashboard is colorful with graphs plotting numerous KPIs with little guidance to operations. But this doesn’t tell you what is or isn’t impacting the business, nor does it give you a triaged list of what is most critical at any given moment. You can have many colorful counters and alarms, but they’re useless if they don’t identify an order of importance to your issues, provide proactive insights and guidance, and tell you how to troubleshoot the issues.

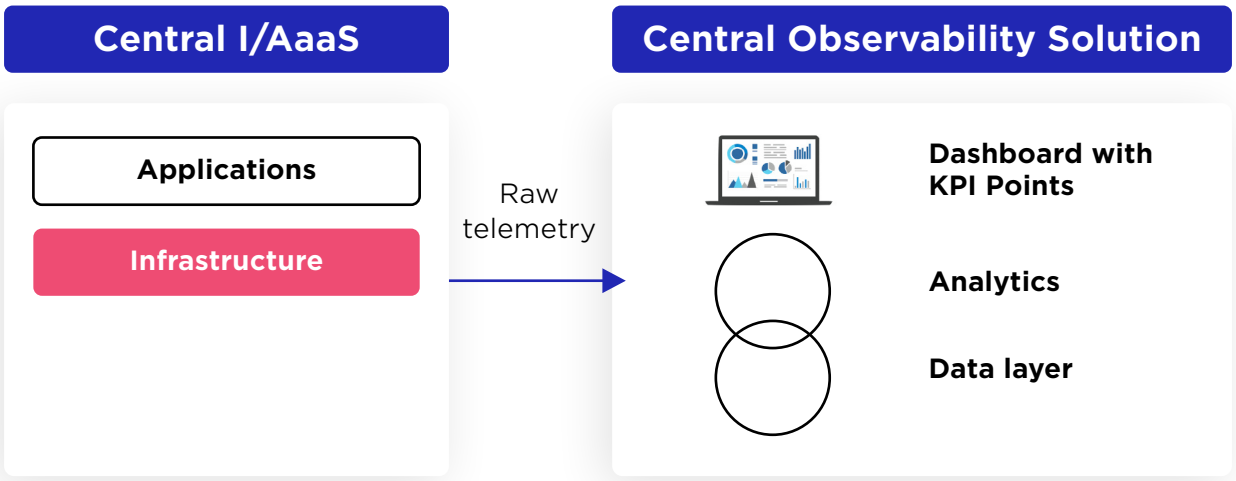
Muddying the waters with a discussion about new terms such as observability versus monitoring runs the risk of validating this naive approach.

For example, observability is defined by some as simply collecting metrics, event logs, and traces (whatever we have) and adding analytics on top of that. This overly simplistic view will not give you an observability system that improves your solution.

Signs of the naive approach:

1. Users report most of the issues, not the monitoring solution.
2. Too much focus on managing the data platform.
3. Monitoring dashboard does not give proactive guidance on where there are issues.
4. Hard to analyze underlying reasons for issues.

Naive approach



The scientific approach

Scientists do not just randomly look at their measurement instruments and see answers pop out at them. They define research goals and measure what is needed to answer their specific questions. The same approach should be valid for observability.

We need a way to ask questions about the health of our systems, and observability should provide answers and detailed support to help troubleshoot the underlying issue.

It is also important to base the observability system on some facts:

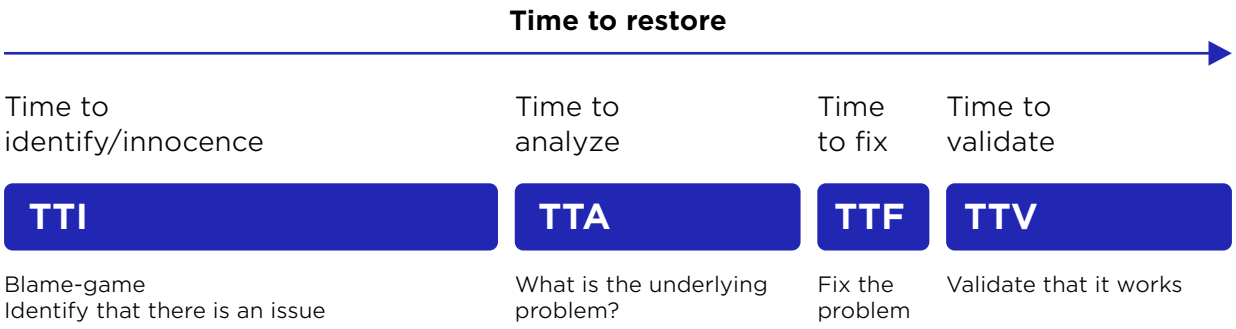
- **Many issues are not faults/blackouts rather performance issues/brownouts.**
- **Many issues are related to non-optimal or erroneous configuration issues, not failures.**

Therefore a sound observability system must:

- **Tell *if* the system works and *why* it is not working.**
- **Focus on application *health* and not just failure and blackout scenarios.**
- **Be defined by the valuable insights it gives rather than a collection of logs and metrics.**

Google has put together an excellent summary of relevant observability principles in a section on **monitoring distributed systems**. It summarizes principles to reach the goals outlined above.

A proper observability solution should support all phases of a resolution process, which is illustrated below:



Time to identify/innocence
This is the most costly phase in many situations. Users have to convince the operations team that there is an issue. In many cases monitoring solutions do not detect that there is an issue.

Time to analyze
Once we identify that there is an issue, we need to be able to understand what the problem is.

Time to fix
The problem needs to be fixed. This can mean changing the configuration or restarting a container.

Time to validate
Sometimes overlooked, after a fix, we need to validate that the health is restored until pinging the customers or users.

How can we build an observability system that supports the complete resolution process as outlined in the previous page?

The four observability pillars

Before moving onto the specifics of edge observability let's dig into pillars needed for a good observability solution:

1. **Bottom-up pillars**

   The most often talked about include metrics, event logs, and traces. In the naive approach, all focus lies here.

2. **Top-down pillars**

   These, including synthetic measurements and probes, are sometimes overlooked.

3. **Bridging pillars**

   These connect various measurements into something meaningful.

4. **Action pillars**

   These include tools to perform troubleshooting and restore health.

Bottom-up pillars

The bottom-up pillars are primarily concerned with white-box data produced by the individual components: metrics, event logs, and traces.

These data sources are most valuable in the **time to analyze** phase when you want to understand what went wrong. They are less useful in the **time to identify** phase, when you're trying to proactively detect if something is wrong. It is often too challenging to pinpoint which fraction of the events and metrics point to a system's health degradation. Also, many degradations issues are due to misconfigurations which often do not result in events. In addition, it is important to consider the challenges of data quality and historical efforts of filtering and correlation. To some degree, modern technologies like anomaly detection can help the situation.

Developers with an observability mindset can assist with improving data quality in the bottom-up pillar.

No code is done until you've built instrumentation to support it. Without building in this kind of visibility, operations teams cannot determine why a system has failed. You need to consider the users of the data, what is needed to troubleshoot, and which events are needed for the complete resolution process.

Top-down pillars

Bottom-up pillars are crucial to understanding how to troubleshoot individual components. However, they are less valuable when trying to understand the system's overall health.

Therefore we should balance our solutions by adding more top-down measurements. And the most important measurements are results from *synthetic probes* and *health states.*

Synthetic probes act as clients or end-users.

They generate requests to the application and validate the response compared to what is expected. This can be HTTPS or TCP requests, for example. The tactical thing here is that this proactively detects issues impacting clients. It also detects complex issues that correspond to a misconfiguration, which normally is hard to detect from events and logs. It is tough to improve the **time to identify** without using synthetic probes. Synthetic probes can act towards components or the system as a whole. In the latter case, the probe directs traffic towards the endpoint that clients consume.

Another important top-down strategy is to implement a consistent *health state* across components.

Each component should have a *few* overall health states. This might be locally inferred from the bottom-up pillars. But the important thing is that developers should provide the observable aggregated health state as part of the component/application. This should not be left to the operations team to infer from a large amount of metrics, event logs, and traces. Leaving that to the external observer can require them to invent the wheel again. The developer of the component has the context and domain knowledge for providing that information.

Bridging pillars

With the above bottom-up and top-down pillars, we have data sources for building a good observability solution. But a fundamental piece is missing; we need bridging pillars to make the information even more useful.

These include:

Meta-data and context

Observability data must be enriched with contextual information such as locations and meaningful names. This is a pre-condition both for human consumption and automation.

Dependency maps, topology

Individual components are most often part of a dependency tree; containers belong together in an application, hosts are connected via an application network.

State aggregation

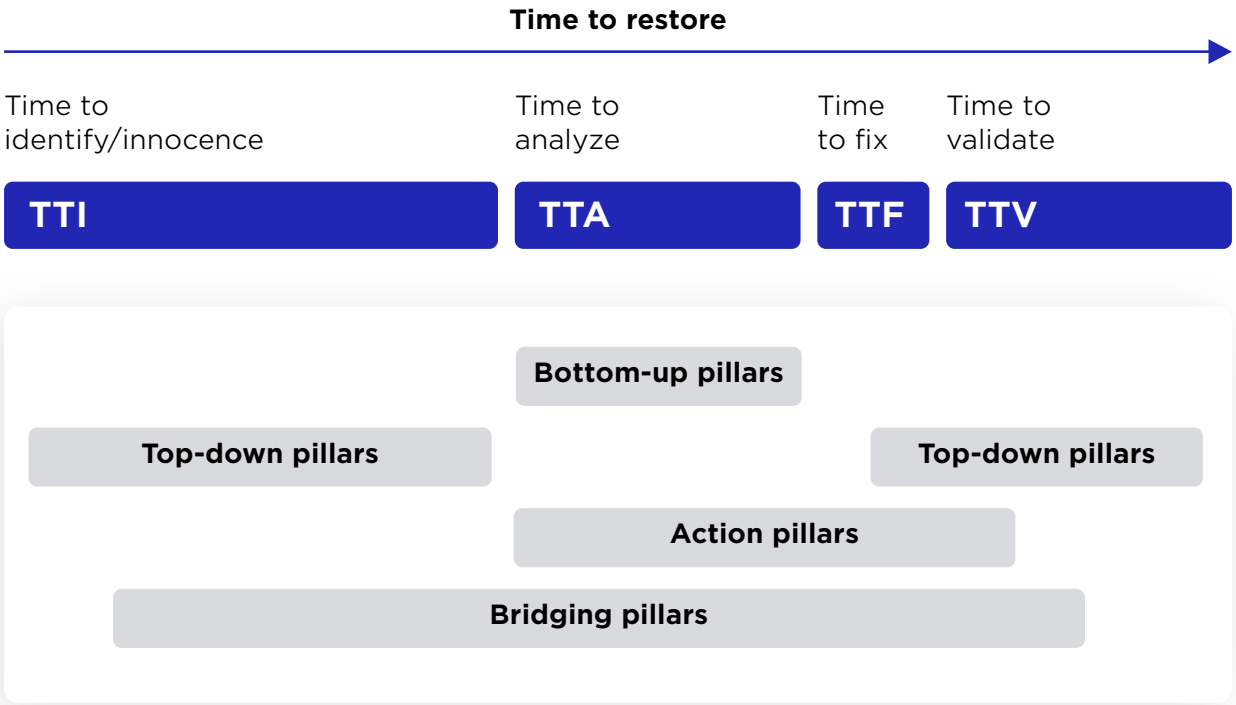
Using the items above, states can be aggregated to show an overall health state for a site, an application, or a type of service.

Action pillars

To be able to troubleshoot and fix the problem quickly, the observability solution can not only be a read-only system.

We need to optimize the entire **time to restore** process, which involves troubleshooting tools and fixing the problem. Therefore, the observability solution needs to embed proper tools to dig into components, restart components, run self-tests, and reconfigure applications easily. Without this, there is a risk that there will be information loss between operations roles and systems, resulting in a longer **time to restore**.

With this toolbox at hand, we can address the complete time to restore process as outlined below.



03 What is different with edge observability?

Before digging into how we can specialize the observability solution for the distributed edge, let us summarize how that differs from a centralized IT solution.

	Centralized IT	Distributed Edge
Placement	The applications are centralized in a few data-centers	The applications are running on a large set of distributed sites
Connectivity	Central data-centers can be assumed to always have network connection	Edge sites can be disconnected from the network
Centralized or distributed data management	Bottom-up pillars can constantly be fed into a centralized data analytics solution	Edge sites might be disconnected and the amount of data from each site might be too much to send to a centralized solution
Infrastructure homogeneity	Control over homogenous infrastructure makes it easier to control application and infrastructure dependencies	Loosely coupled to heterogenous infrastructure makes the blame game more complex between application and infrastructure teams
Centralized or distributed tools	All tools/pillars can be centralized	Pillars/tools need to work towards, and at, the edges

Context	Physical data centers do not have contextual meaning in the observability process	Edge sites and physical location have contextual meaning
Deployment handover to operations	Application deployment time span is short. Since few application instances are installed in central data centers with high-quality networks the deployment phase for each deployment is short	Application deployment time span can be long due to the number of sites, slow networks, network outages, etc.

Summarizing the table above, we see that an edge observability solution can not just send bottom-up data to a centralized data analytics platform.

There needs to be site-local health and metrics that are filtered and aggregated before sending it to the central solution. It also needs to provide perspectives on the infrastructure and the applications due to the heterogeneous environment.

The edge observability solution also needs to manage the combination of centralized observability with distributed sites and applications.

The tools and views need to be centralized in order to support an efficient operations process, you need centralized insights. On the other hand, when troubleshooting and fixing, you need simple ways of directly reaching the edges.

04 Six solution principles for edge observability

Below follows six design principles for edge observability. They are illustrated with some example screenshots from the Avassa system and an imaginary popcorn manager application at each edge theatre.

Principle 1: Application centricity

It is essential to focus on what matters.

Edge exists to run localized applications. It is the health of the application that matters.

It is easy to fall into a trap where the observability focus lies on the edge orchestration platform itself or with the infrastructure. This will lead to a situation where users detect application issues before the observability system.

The solution needs to be able to see the status of the application across all edge sites.

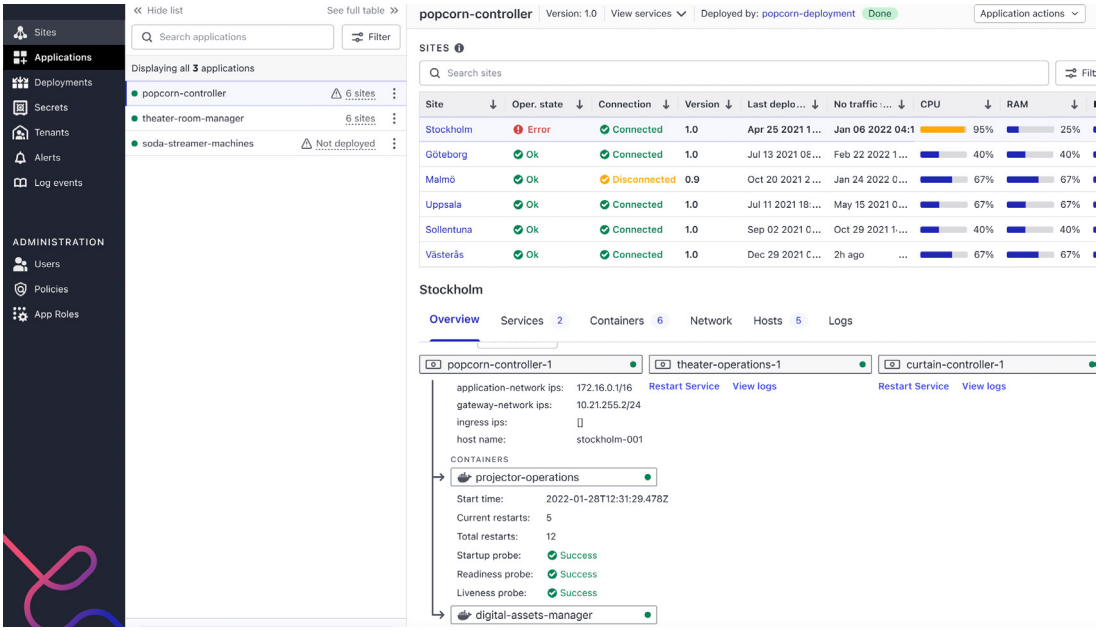
The top-down pillar is central to address this principle.

Principle 2: Edge site awareness

Applications run on sites. Sites are fundamental for the edge use case.

If you own the application, you are most interested in knowing the application's health at each edge site.

The screenshot below illustrates application centricity and edge site awareness. You can select an application and understand its health on each site. Note that you can also see the overall health state, “Oper. state”, the status of synthetic probes, have direct access to the containers, and see all associated logs.



This illustrates the use of all four pillars at the edge site

1. We can reach logs for the applications and containers at a specific edge site.

2. We have overall health state and synthetic probes for the applications on the edge site.

3. We have bridging pillars, including the application and container structure, dependencies between them, and site information.

4. Actions are available to interact with the containers directly on each site.

Principle 3: Deploy and operations perspectives

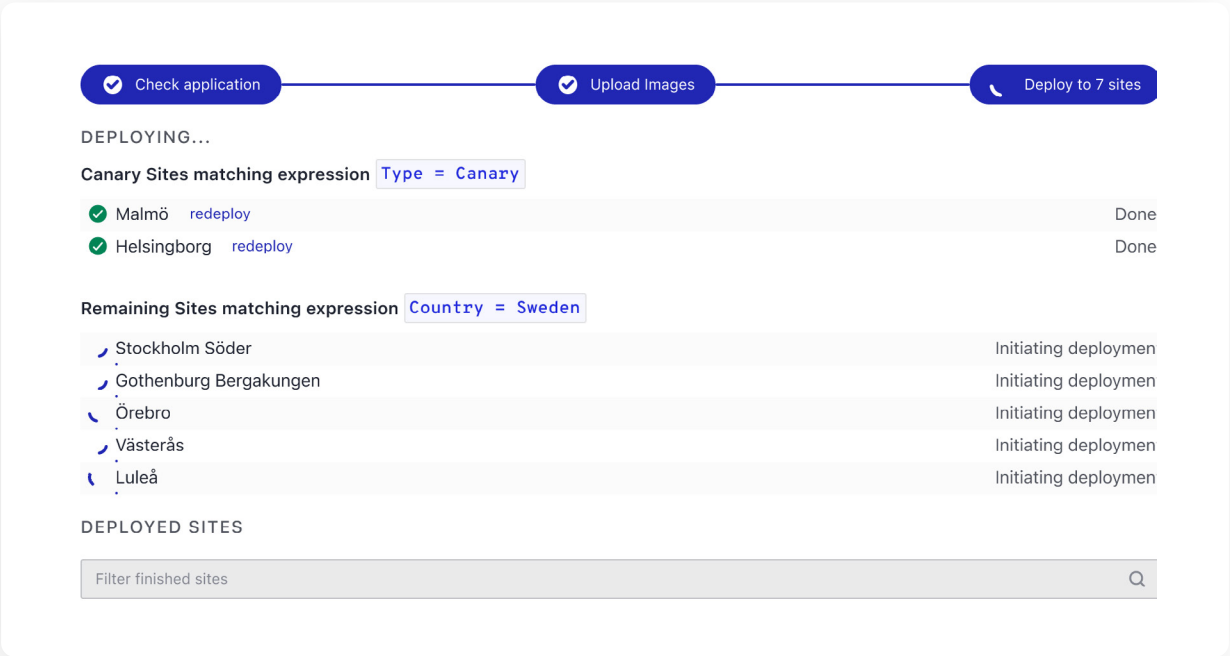
When deploying or updating an application to thousands of sites, the solution needs to provide good insights into the deployment process itself.

This is because the deployment process for edge is more complex than it is for central clouds. The time span is much longer; waiting states or failures states might span days. An edge deployment might need to wait for network connectivity to an edge site, for example.

It is possible to have 80% of the sites with running applications that are moving into the operations phase, while 20% of the applications are not yet healthy after deployment.

This must be visible so that daily operations is aware that the 20% aren't ready and should be sent them back to the delivery team. Applications that are not yet successfully deployed should not be the target for ongoing monitoring. They are still in the deploy phase. It is also essential to support drill-down functions so the outstanding steps in the deployment are visible.

The screenshot below illustrates insights into a canary deployment across edge sites. Both deployment and operations teams need to understand that specific sites are up and running where others are waiting for the canary sites.



This addresses a well-known handover problem between delivery teams and operations teams. Did the application and services even work at delivery or is the operations team chasing delivery problems? This issue is more challenging in the edge use case and must be well addressed.

Principle 4: Infrastructure and application views

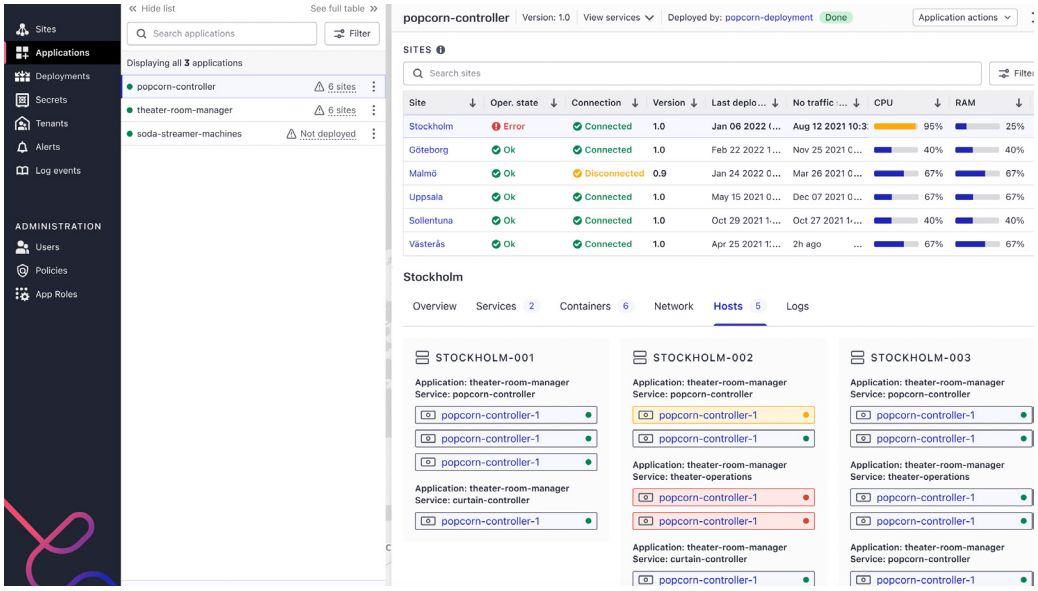
For edge, the application and infrastructure layer are more disconnected than in the central cloud case.

This creates even bigger communication challenges between the infrastructure and the application teams than in traditional central observability. As a result of this, the **time to identify** is extended in a negative way: *where is the issue?*

The observability solution must therefore give full insight into the underlying edge platforms as well as the application layer and the dependencies between them.

In many cases, the infrastructure is owned by one organization and the applications by another. This kind of multi-tenancy must be well supported and at the same time make sure information can not be shared between tenants for security reasons. In addition, there must be enough information provided to speed up any issues. For example, application teams need insights into the to compute resources, and the infrastructure owner typically needs insights into which applications are running on which edge sites.

The screenshot below illustrates observability insights into the edge sites and compute hosts.



It also clarifies the mapping between the applications and where they are running. This means that the infrastructure team will know which application containers are running on which hosts. And similarly, the application team will see which containers are running on which hosts. This is critical in speeding up the resolution process

Principle 5: All four pillars combined

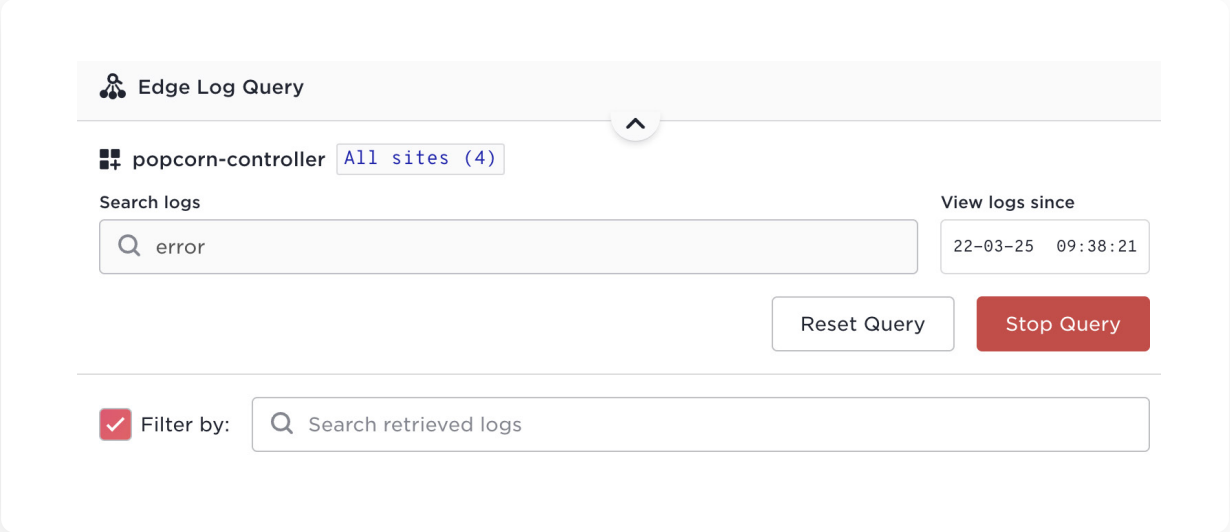
As briefly summarized above we need all four pillars applied locally to each edge site. This is elaborated here:

1. Bottom-up pillars

Not all logs and metrics can be streamed to a central application. The edge site should filter, analyze, and store the data locally. Local troubleshooting must be supported if the connectivity to the central site is down. Also, in large-scale scenarios, it will be too much to send everything. A subset and aggregated data set should be forwarded to the central solution.

Furthermore, the central solution should support distributed search mechanisms to dig down into the sites when needed.

The screen shot below illustrates a distributed log search across sites:



2. Top-down pillars

Synthetic probes should be run at each application site to determine the externally visible health. Use of mechanisms like startup, readiness, and liveness probes are strongly recommended. Furthermore, the overall health state should be aggregated per edge site and per application per edge site.

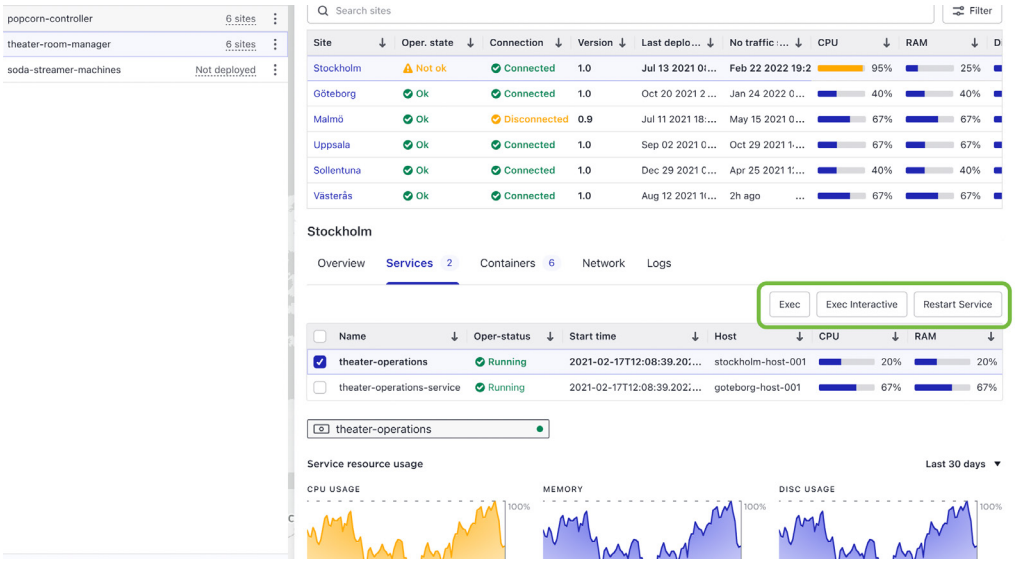
3. Bridging pillars

Metadata like application information, application and container structure, and site data such as location is fundamental for the observability processes. Human operators need this to understand where the issues happen, and automation tools need this to take action.

4. Action pillars

In the distributed edge case, operations teams need centralized tools to quickly dig down into a specific container in a particular site. This should not assume manual steps and look up information manually. As an example, if a problem seems related to a specific container an operator should have direct access to tools to manage and inspect the container. It should not be necessary to manually find out addresses, hosts names and login information, and run container commands manually.

The example screenshot below illustrates how an operations user has drilled down to an application on a certain site and can perform container operations such as restart and exec:



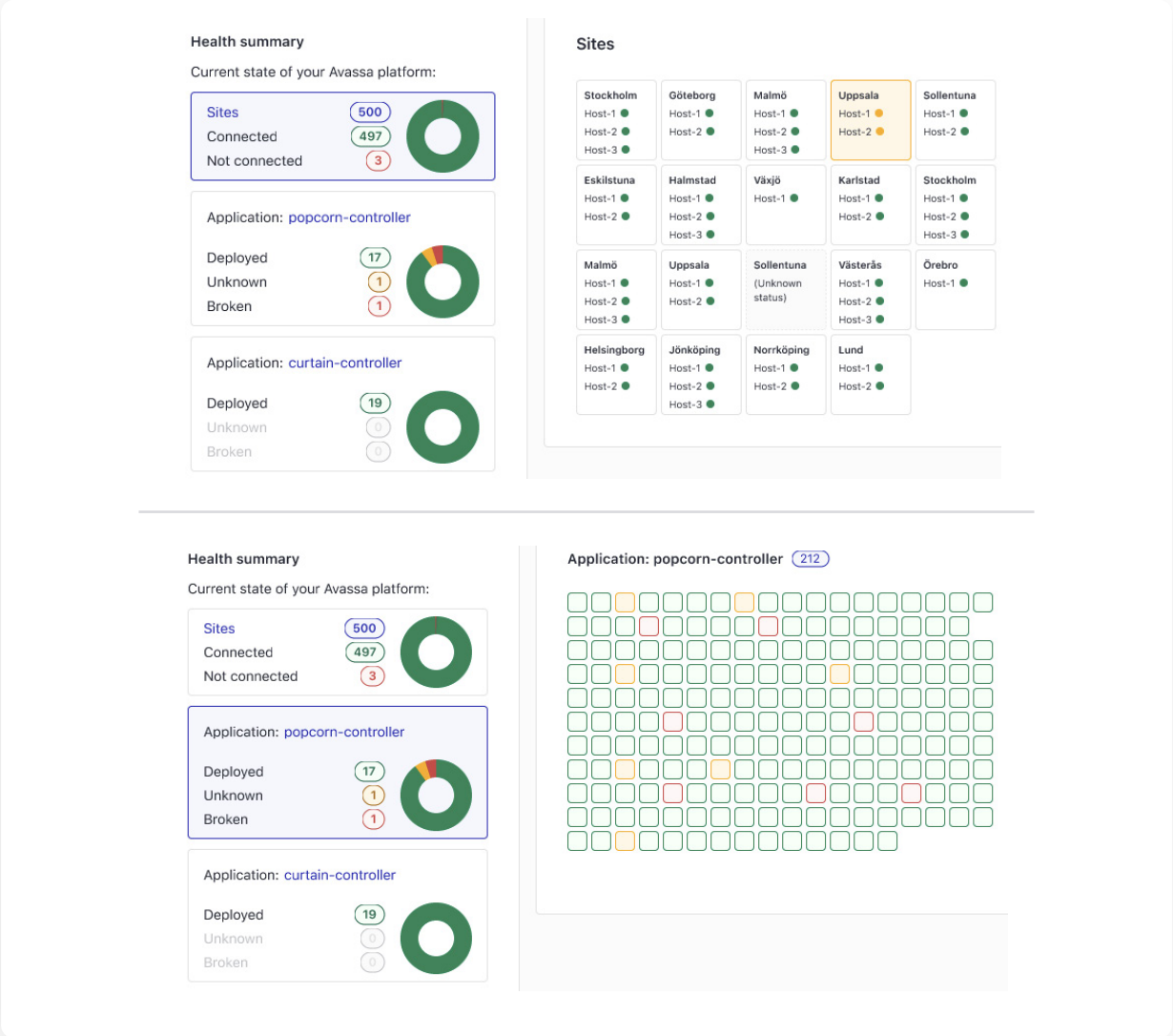
Principle 6: Managing scale

Each scenario can include thousands of sites with numerous applications on each site.

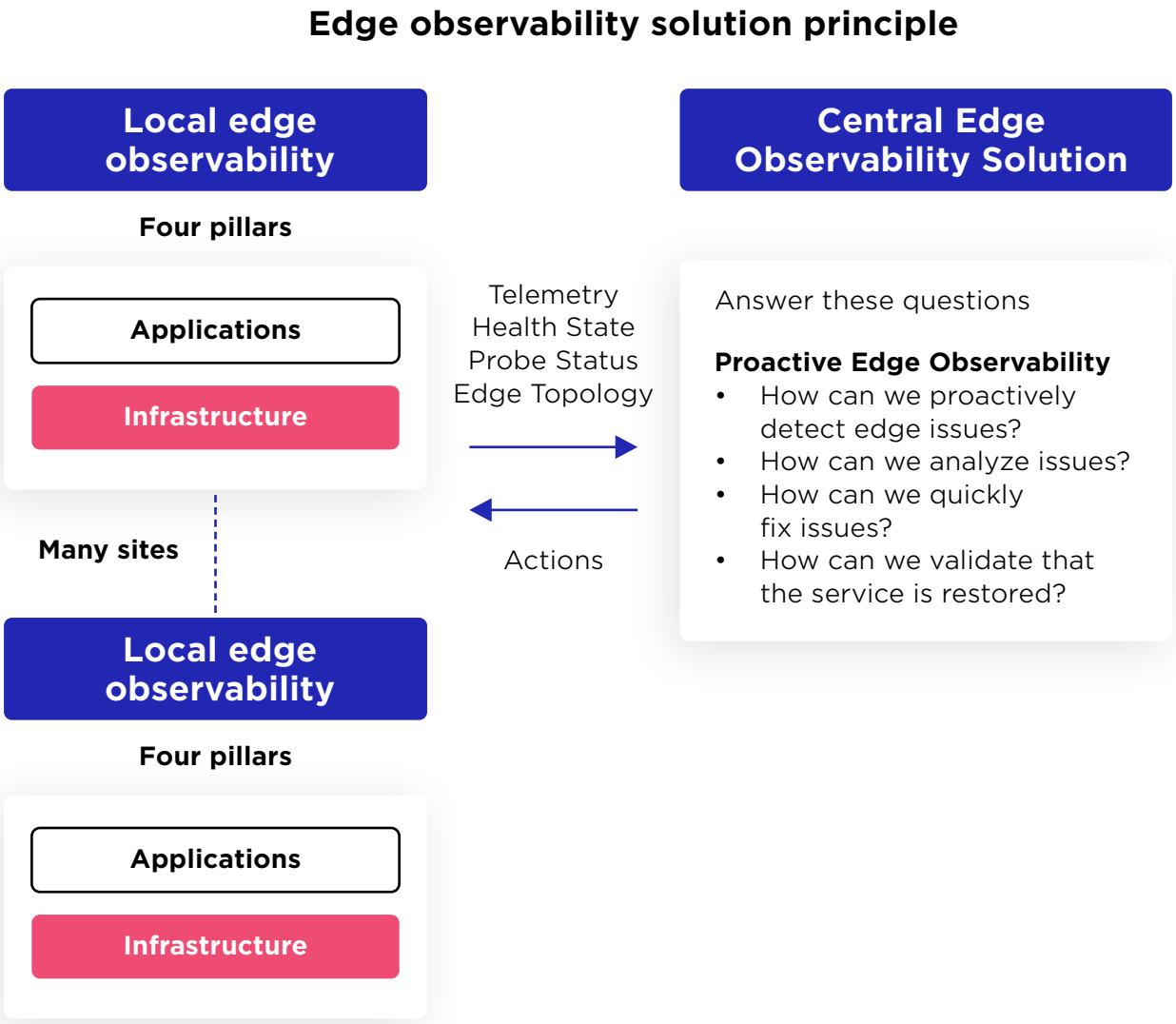
Views and tools must be designed for this. Operators need views that can present this in an overview but include quick drill-down to details.

Managing metrics, logs, and more in the edge use case will also put scaling challenges on the solution, it is probably not feasible to forward all low level data from each edge to the central application. As stated previously, proper division with what is stored locally at each site and forwarded to the central solution must be balanced.

The screenshots below illustrate the use of heat maps as one technique to give an overview of a large number of sites and applications:



The illustration below illustrates the principles for the edge use case:



All four principles are applied locally at each edge. The interface between the edges are not just telemetry data, rather an interface supporting all pillars. This will result in a centralized edge observability that answers the fundamental questions for the complete resolution process.

05 Conclusion

Do not forget to build the edge observability solution around which questions need to get answered rather than focusing too much on the data platform technology.

Make sure to apply the four observability pillars in order to support the complete resolution process.

And **let application focus and edge awareness guide the design.**