SOLSYS

CONVERGENCE OF OBSERVABILITY & SECURITY:

The Benefits and Approaches to Better and Safer Applications and Service

CONVERGENCE OF OBSERVABILITY & SECURITY:

The Benefits and Approaches to Better and Safer Applications and Services

The more you can see, the more secure your ecosystem becomes. Observability and security intertwine to create a more secure environment.

Security exposures and compromises happen in the dark when no one is watching. Applications with good observability, however, are inherently watched. Implementing instrumentation to ensure an application is running was crucial in order to observe possible security issues.

Modern applications have become more complex, leveraging more containers and systems in the cloud while integrating more libraries, services, and backends than ever before. This complexity is evident for small business products and enterprise applications alike.

What does this mean for cybersecurity? Security exposures are more likely than ever to exist. Running complex applications and keeping them performant and reliable, requires in-depth visibility into what is happening in the stack of code and tools you're running.

Instrumenting applications and containers to consider both observability and security gives us a broader set of events with more comprehensive information to consider, supporting the reliability and security of our products.

What is Software Observability?

What exactly is observability? Let's review the essential categories of precisely what we mean when discussing software observability and the information we should capture.

Capacity Metrics

These metrics provide visibility into container and server resources used by the application — traditional measures such as memory, CPU, and disk usage. On top of these, we can talk about thread or container usage for applications that automatically scale up and down.

Container, Platform, & Runtime Monitoring

Observability relates to gathering information about containers; anything from the JVM or runtime engine to the application container or gateway (such as Tomcat, JBoss, Kong, etc.) and to Docker or other types of server containers where your code may run. Like capacity monitoring, these metrics are not always application-specific but will tell you about how the container itself and your code or deployed application is interacting with it.

Database observability falls into this category, where the database reports information like read and write performance, storage, and search times. These metrics allow you to see how the application behaves with regard to the hosting containers and the libraries with which it interacts.

Standard server or virtual machine logging also falls into this category. Servers and VMs output volumes of information about capacity resources and details about connections, logins, and other interactions.

Audit Logging

Understanding application usage gives valuable insights into application transactions. For example, knowing that an application had performed a task, who it was for, when it occurred, who authorized it, and the impacts (success or failure) of an event is valuable information.

This kind of observability is usually highly application-specific, though even web servers deliver audit logs for each HTTP transaction, for example. These logs usually contain essential summary information about a 'business event' and can help you understand when transactions fail and why.

Performance Logging

Understanding how long your application takes to perform specific tasks or actions can help diagnose problems in the code base or identify customers using your application in unexpected ways. This data is often logged with audit transactions or can be gleaned from container monitoring.

This data is extremely valuable, as it helps understand whether your application is delivering to customers promptly and consistently or if your customers are being left waiting and walking away from valuable business interactions.

Dev/Ops Tracking

Any time an application runs into a restart, resource shortage, connectivity issues, or error conditions, you need to know about it. Logging this information will enable DevOps teams to determine where the code might require improvements or what might have happened regarding dependencies or resources that could have triggered outages or errors - even if your application automatically recovers or asks the customer to try again.

Access logging can also help with this kind of insight, where access logs show requests to the application before they are completed. Completing a request is often where applications run into resource or software defect issues, so gaps between access and audit logging can help identify issues and 'lost operations.' Additionally, many of the libraries or components you are using may output data to log files in the event of issues, misuse, or even just with day-to-day use.

Business Tracking

Going beyond audit logging, we can enrich the types of information we're capturing and tracking business metrics and events. For example, completed transactions, perhaps with crucial anonymized summary metrics such as the product sold, the total value of the transaction, or any other metrics related to key business objectives. These logs allow for quick and easy reporting of high-level business performance and operations of an application or service.

Not every type of observability data is appropriate for every application.

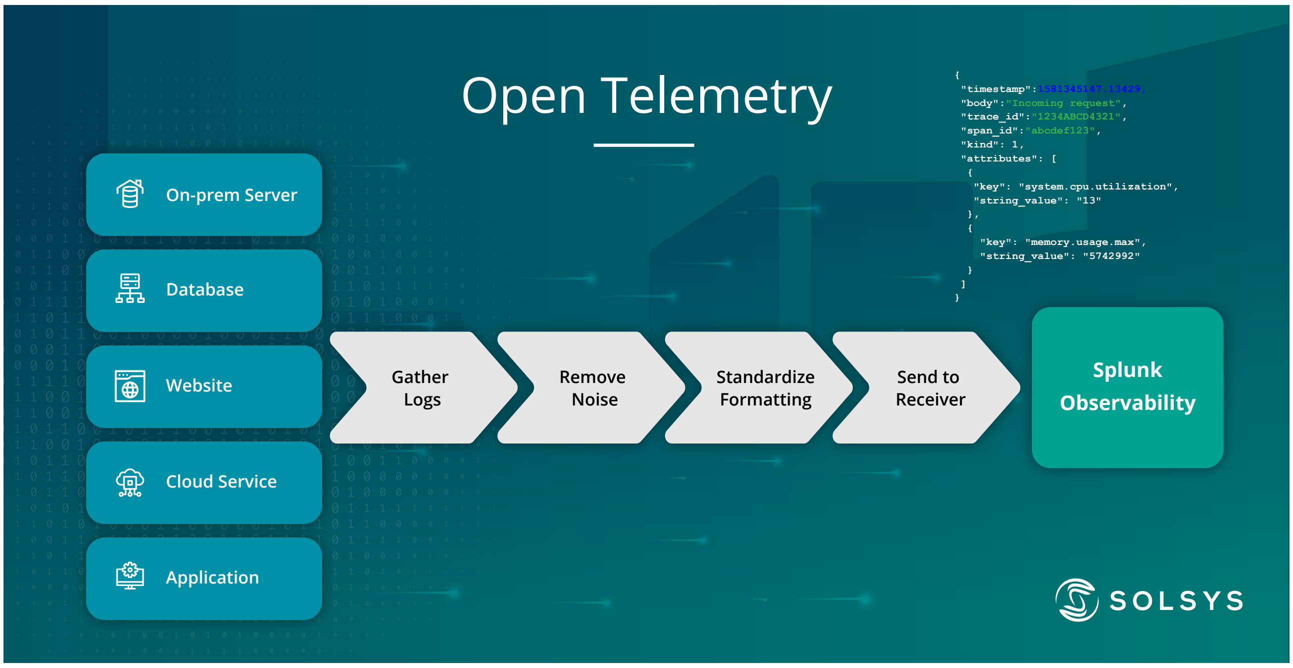
You can often combine certain types into single log entries (such as rich audit logs). You don't need to log all of this data. Still, aggregating available data and feeding it into a centralized location that is observable by team members and stakeholders helps make your applications visible — providing deep insights into applications over time.

How Do We **Gather** & **Record** This Information — And Where Does it Come From?

Tools like Splunk Enterprise and Splunk Observability can help automatically gather this information and put it into one place. These platforms use open standards (like OpenTelemetry, SNMP, Syslog, etc.) or even custom agents — lightweight components installed on servers to capture and forward log files, run commands, or execute custom code that knows how to communicate with your infrastructure or web services to capture data.

Additionally, various cloud platforms offer services such as AWS Cloudwatch and Kinesis to capture and feed information from your cloud environment to a centralized logging application.

Custom logs are a powerful way for your application to capture and report information. Some of your logs will get 'spit out' from libraries or containers, but having separate well-groomed custom logging — whether to a file, database, or elsewhere — will mean you can capture specific events with details pertinent to your application. Let's explore how we make these logs most effective for observability and security.



```
{
 "timestamp":1581345147.13429,
 "body":"Incoming request",
 "trace_id":"1234ABCD4321",
 "span_id":"abcdef123",
 "kind": 1,
 "attributes": [
  {
   "key": "system.cpu.utilization",
   "string_value": "13"
  },
  {
    "key": "memory.usage.max",
    "string_value": "5742992"
  }
 ]
}
```

It's worth grabbing as much data as possible from all relevant components, regardless of how you configure your application. Ensure you're capturing logs and data from all infrastructure components that are part of your product, such as:

- Servers and hosted services
- Containers and runtime environments (docker, JVMs, etc.)
- Gateways and proxies (Kong, load balancers)
- Cloud infrastructure and services
- Application libraries and components outside the custom code or core software
- Dependant services - response information can often be captured and logged
- Network infrastructure and even raw packet captures, if possible

You don't have to capture or create perfect custom logging on day one of deploying your application or service. This information can be improved over time, just like your product offering to customers. Ensure your product owner and stakeholders understand the value of investing some time into occasionally improving the logging and observability of your application.

Gradually improve the richness and accuracy of custom logging as you touch parts of your code that can add capabilities like audit events. As you improve or change infrastructure, spend a little extra time to ensure it is instrumented and reporting information whenever possible. For example, adding packet capturing to key network segments might be something you introduce later if and when you are trying to resolve or investigate specific performance issues between components.

Taking a Big Data approach to this type of data aggregation is essential. A Big Data approach means you take the observability data as is, even as it changes over time, and can add structure and reporting on it later as you might need it. Gather all information into a single place organized to understand where it came from. Splunk is an ideal tool for this, but many products are available to help.

Keeping capacity and operational types of metrics and logs longer than a few months might not be worthwhile, but storing audit-related data for a year or more can be extremely valuable for trending information in terms of usage. It also helps with security reporting; we'll discuss exactly how below.

What Are the **Benefits** of a Well-Observed Application?

8,000+
Active Application

3,400+
Peak Transactions Per/Sec

12B+
Transactions Annually

10,000+
Automated Tests

95%
Reduction in MTTR

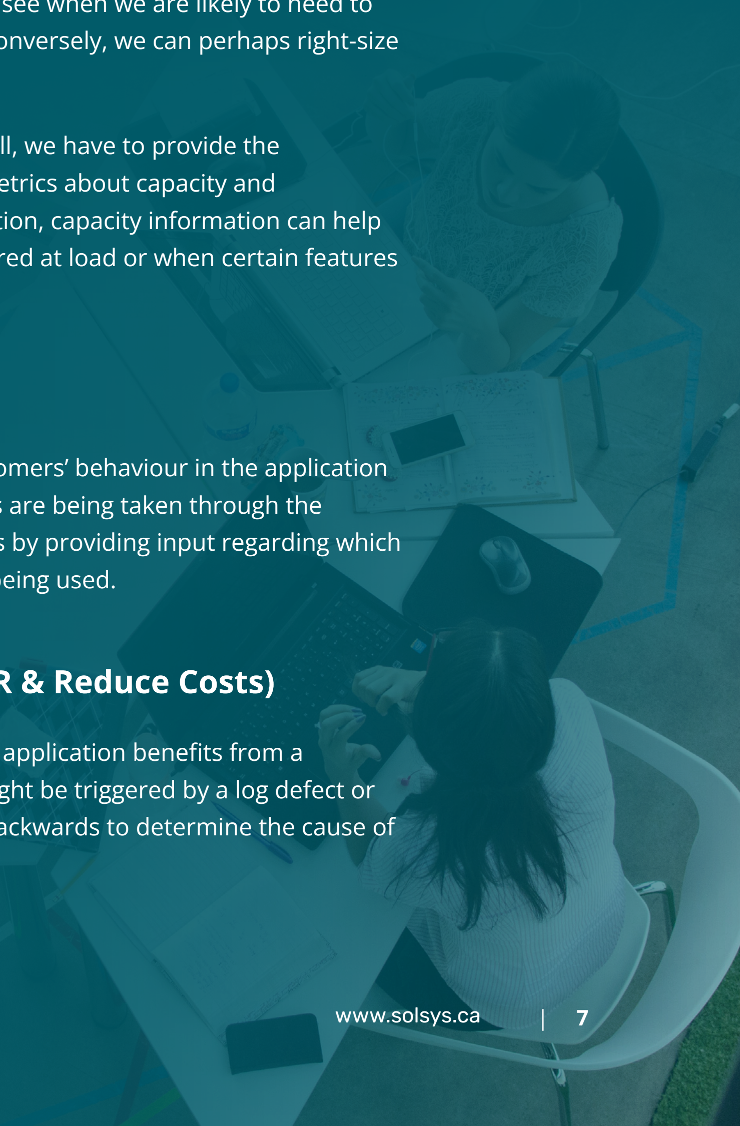
0.0004%
Error Rate

99.999%
Reliability

75%
Reuse

A well-observed application can be better operated by a DevOps team because they know how to review and interpret the output. Good observability can help with many important areas including:

- Plan for growth from a capacity perspective.
- Understand how our application is being used and who is using it.
- Resolve defects or find issues with the application.
- Ensure we're meeting our SLA and the application is performing well.
- Avoid outages or resolve them quickly.
- Report on business metrics helpful to stakeholders.
- Reduce costs by reducing Mean Time To Resolution (MTTR) of issues, and ensure we're not over-provisioned in computing resources.
- Increase customer satisfaction by ensuring performance and fewer defects before our customers tell us about them.



Plan for Growth

Capacity metrics show us how our infrastructure is performing. Correlating customer traffic with resources (vCPUs, disk, memory, containers, etc.), we can see when we are likely to need to ensure our system is scaling by investing in more resources. Conversely, we can perhaps right-size deployment resources to save money.

With cloud computing, this scaling can often be automated. Still, we have to provide the parameters for automatic growth (and server reduction), so metrics about capacity and performance can help in any deployment architecture. In addition, capacity information can help identify a symptom of a software problem that could be triggered at load or when certain features have been used.

Plan Products

We can see which features are most used by tracking our customers' behaviour in the application by using anonymous clickstream analytics, seeing what actions are being taken through the software. This capability helps our product teams plan updates by providing input regarding which features are most important to customers and which are not being used.

Support & Debug Applications (Reduce MTTR & Reduce Costs)

Logging has always been used to find defects. A well-observed application benefits from a complete picture of the application to help determine what might be triggered by a log defect or an unexpected crash or failure. Enabling developers to work backwards to determine the cause of an error is extremely helpful.

Are we reaching an issue because we have run out of computing resources? Is that because we are experiencing a higher load than usual? Was a customer accessing or using a new or rarely used product feature? What was happening before the defect occurred? What customer data was involved? A well-observed application can answer these questions thoroughly.

When customers inevitably experience an issue and engage our support team, resolving their issue quickly is vital. Not only does this improve our customers' experiences, but it also reduces an application's operational costs and may result in better business outcomes and greater profits.

DevOps teams need data to understand what went wrong for a customer. In many cases, our application may not even have a defect. Still, perhaps the customer's specific account or scenario isn't supported, which wasn't made clear to them with the software. Diagnosing the specific scenario, what the customer was trying to do, and how the system responded can help us answer these questions quickly.

Reducing the MTTR doesn't just help customers and solve problems quickly; it also reduces the time our team has to spend on operational support. This reduction represents direct cost savings and gives the DevOps team more time to spend developing new aspects of the product instead of operations and support.

Turning it off and on again is the least satisfying way to support an issue in production. Instead, good observability can help us avoid this, find root causes quickly, and avoid the same issue(s) occurring again. Customers will respond positively to companies that can explain what happened and how they will fix it. Observability puts us in a place where we can do just that.

Measure Our Service Level Agreement (SLA)

Application performance - especially in services - is more critical than ever. Hardware is powerful and available, but we still run into slow applications or products. Through measuring performance, we can report dynamically on our SLA and see where and when we might not be able to meet it or where we can improve.

Synthetic transactions can be extremely valuable for this goal. Logging internal timings of events and processing times are vital and valuable. An external perspective that reports on the customer's experienced time from different regions takes things to the next level.

Good synthetic monitoring services can also break down where time is being taken — is your DNS server sometimes slow? Your customer-facing network? How well is the web page written? Does it take time to render in the browser in a way you didn't anticipate?

Improving Customer Experience

A complete product picture allows us to improve the application's overall quality. We can start to predict when issues might occur; we will know when our high traffic times happen and can plan for them, and we will reduce the time it takes to resolve issues and support the application.

We'll get to a place where we spot defects and repair them before a customer can tell us something is wrong. Reaching this point has a cumulative and net positive effect on customer experience and satisfaction.

Observability — knowing what's going on and being able to report on it — can be a significant way to improve software and help deliver its real benefits to the customer.

Deliver Business Reporting

When we build a product, we want to know how well it's doing. This question can often be answered by seeing how customers engage with the application. Reporting on total logins, transactions, sales, and so on are often key indicators of how things are running. We can use the same instrumentation and information we use to make the application supportable to report on business performance.

Similarly, if we add an application log to help with security, we can also add a few bits of information to get basic business reporting from the same content. Telling a story about our product and its usage is vital to the business and it can also be inspiring to our DevOps teams as they see how their products are being used by actual customers.

Improve Security Posture?

The above benefits alone make instrumenting and setting up for observability well worth the effort. However, if we do this instrumentation well, we can also add an additional benefit, or at least set ourselves up to benefit from it: **an improved application security posture**.

So, let's dive into some more specific details about how we can make this happen.

Security with Respect to Observability

How is observability going to help us watch for security events? Capacity monitoring isn't going to reveal a lot about possible security exposures. Just because an application is using a bit more memory than usual, it doesn't mean someone has compromised the application stack (unless perhaps traffic hasn't increased). Yet, almost anything in DevOps events, audit logging, or business tracking domain can help improve the security posture.

With audit, business, and operations logging, we're looking into some key types of events:

- Who is doing what, and when — such as authenticating, being authorized, or making changes to data or configuration.
- Unexpected crashes, defects, or operational activities that aren't normally executed.
- How many types of events have occurred?
- What business data has been accessed, by who, and when.

Suppose we ingest these logged or instrumented events into a SIEM platform (such as Splunk ES). In that case, we are already positioned to build custom monitoring use cases for our application or to benefit from global use cases. This is especially true if our application uses centralized (shared) Identity Providers (IDPs), which we will discuss more below.

The audit aspects of stored log data can also be valuable for security investigations. For example, suppose information is exposed from your application. In that case, it should not have been, or perhaps if someone's account for your application was compromised, the audit information in these logs could report what information was accessed during the time of exposure. This knowledge can help us determine if a security breach impact is significant or perhaps so small that it's a non-event. Vital information when reporting to stakeholders, security departments, and ultimately the customers and users who have placed their trust in our products.

What Should We Log & Instrument with Our Observability Events to Help Security?

The more information we can track, the better for improving our security posture. The cost of logging information live in production used to be high and could affect performance. In today's world of cheap disks and processors, creating log entries with well-identified information is trivial, so it's worth erring on the side of detail. There are ways to instrument an application other than log files, such as logging services in the cloud (such as AWS Cloudwatch and Kinesis).

Consider including the following types of information with any logged event (especially audit-style content) and you'll likely be ready to trigger security monitoring in your SEIM:

DETAILED TIMESTAMP: Knowing exactly when events occur helps developers, operation teams, and soc analysts trace events through the system and beyond your application. Include the timezone and millisecond accuracy. RFC 3339 is a great format to use that captures anything anyone might need to know for most applications.

WHO: Which user or system caused the event to occur? You may have some restrictions on logging user names. Good IDPs will provide universal identifiers for users that are either specific to your application or at least not part of a credential.

WHAT: What was the event about, and what operation(s) were actioned? What business objects were affected or involved - which operation was it (CRUD)? For example, if it was a change operation, which critical record was being modified? Again, be aware of PII or PCI information, but log details when possible to trace specifics.

WHERE: Which servers were involved, and where did the request come from or go to? Include IP addresses of the source request (and be careful it's not just your load balancer IP - see the X-Forward-For header!)

DETAILS: Log parameters or details about what occurred. If you're concerned about information being logged, you can trace events in a secure database and reference a correlation ID in the logs. You can also use one-way hash data to be consistently unique and identifiable but not easily reversed. Don't log hashed credentials as they can be brute-forced — and logs are not considered secure. For example, you can also add to your logs database ID columns for your data instead of the specific customer name.

RESULT: Include the outcome of the request or action, success or failure, or perhaps more nuanced information as required.

Try to capture information in well-structured ways. For example, consider how your log files are structured if you're not logging into a data store with JSON or XML documents for events or an SQL database.

Try to use something like name-value pairs for data. This method means these 'fields' can be easily found and reported on later, even after they might change names or even disappear from entries. Dumping values into a line separated by spaces makes later parsing and evaluation harder, but even this type of content can be used if required.

Here's an example of a decent 'audit' entry line for a log file in an application that has enabled the creation of an account object:

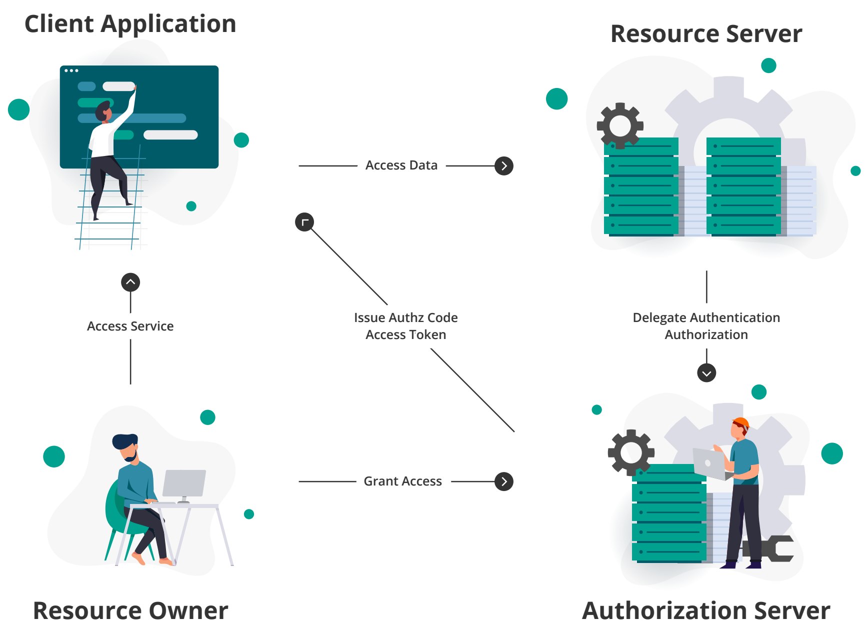
```
2022-09-19T16:39:57-05:00 action="create" type="account" src_ip=1.2.3.4
user="aef6b257-c59c-4f79-b72b-854b571c12b3" created_account="02bd9dad-a8fe-
4642-b78d-efa2b947e350" result="success" db_ms="46" total_ms="59" perm_
lvl="account_admin"
```

Notice how some performance details are captured along with details of who created the account, the result of the action, and the permission levels used to make it happen. Name-value fields separated with = and the value delimited by double quotes make it easy to read for both people and software.

A Simple Example

Imagine our application uses OAuth2 to authorize customer access to our application. If we log authorization events, noting the validity of a token, who it represents, and where the token came from (IP), we can trigger security events where unauthorized users are trying to access applications.

We could also immediately trigger alerts if OAuth2 tokens are being presented to our application regularly that are not valid; perhaps they were never issued by our IDP. This kind of security attack is expected. Thanks to observability, we can push this information to a SEIM that can appropriately alert the SOC that something is happening that may suggest a cyber attack or indicate an issue with the application.



SOLSYS

What If I Need Help?

Solsys has experience helping product teams instrument their applications and turn ‘dark’ applications and infrastructures into well-observed systems. For example, we’ve helped corporations pull data into platforms like Splunk and Splunk Enterprise Security to trigger use case reporting of possible exposures.

We have also helped a major Canadian telecommunications enterprise that was having issues with intermittent visibility. Thanks to Solsys, the company was equipped with complete observability into each aspect we discussed above. Now, the company’s enhanced observability gives them insights at the click of a mouse — generating a year’s worth of insights in seconds. Ultimately, Solsys took them from darkness to light.

Could your business benefit from moving from darkness to light? We’d be happy to help you better understand the value of observability and how it helps improve your security posture with your applications and services!

Contact us at **hello@solsys.ca** today to speak to our consultants!

CONTACT US