

# Application Monitoring Landscape

---



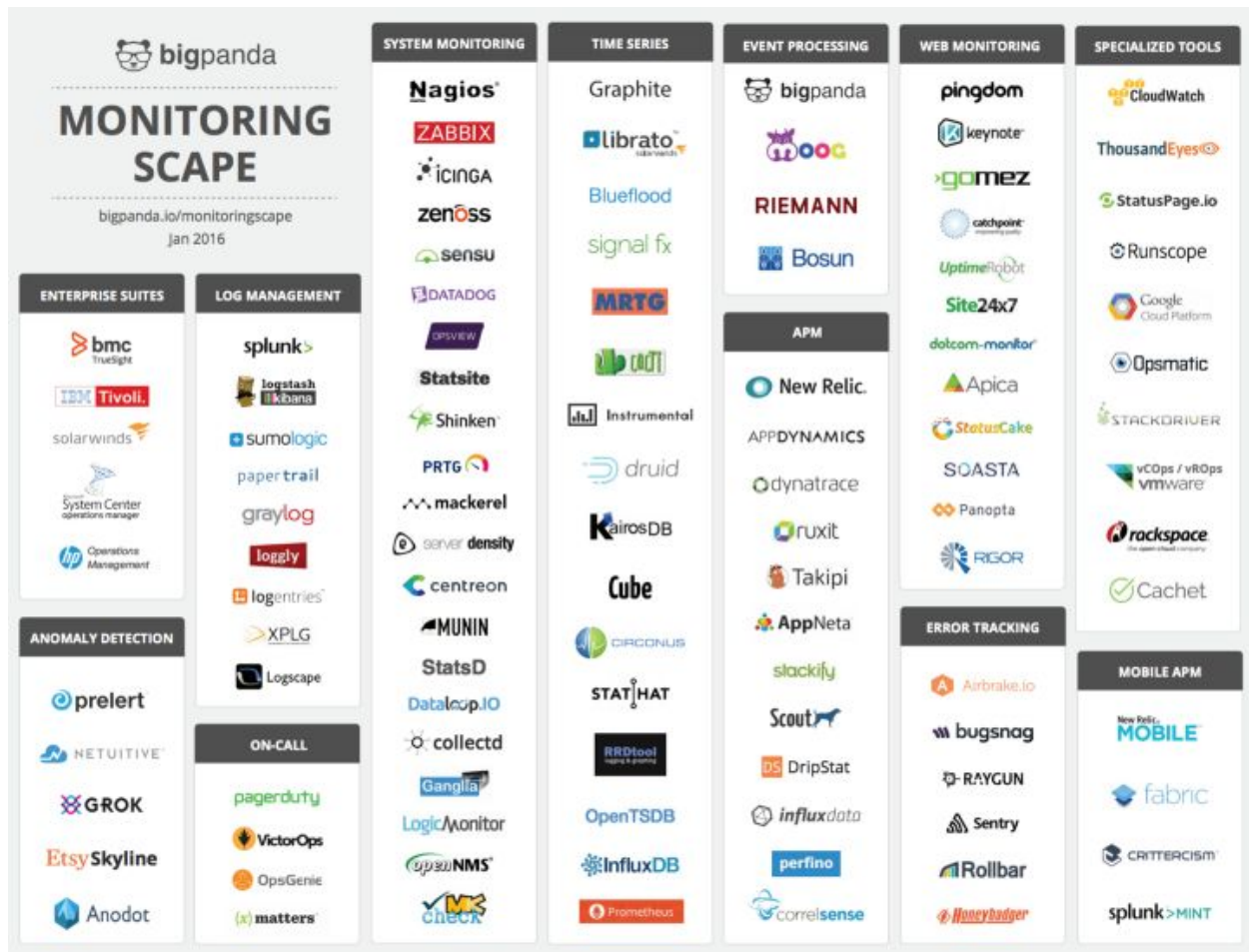
---

## Introduction

Application monitoring space has been experiencing major renaissance in recent years. As illustrated in the popular graphic from BigPanda, there has been an explosion in the number of monitoring techniques, products and associated companies. In this whitepaper, we will describe a framework that we believe will be very helpful in understanding the fundamental techniques and value offered by various application monitoring products. But first, let's review some empirical factors that are driving this renaissance:

- Companies in all industry verticals are critically dependent on software; as highlighted by numerous articles such as: [“All industries are quickly becoming tech industries”](#). As a result, there is ever growing demand for better monitoring products that will ensure higher uptime and faster incident response for business critical applications.
- Modern applications are generating exponentially more metrics. For e.g., [Netflix](#) crossed 2 million distinct time-series in 2012, Spotify was already at 100 million time-series in 2015. Compared to the SaaS giants, even if average modern application is generating an order of magnitude less metrics, that's still thousands of time-series and millions of metrics!
- Technical underpinnings of monitoring products have evolved tremendously. [Time-series databases](#), [Linux kernel enhancements](#) and [machine learning](#) are few examples of key technological advancements impacting monitoring products.

With the understanding of strong driving factors for better monitoring products, let's now focus on developing an easier framework to understand the many choices in this market.



Application Monitoring Landscape

## Understanding the Monitoring Landscape

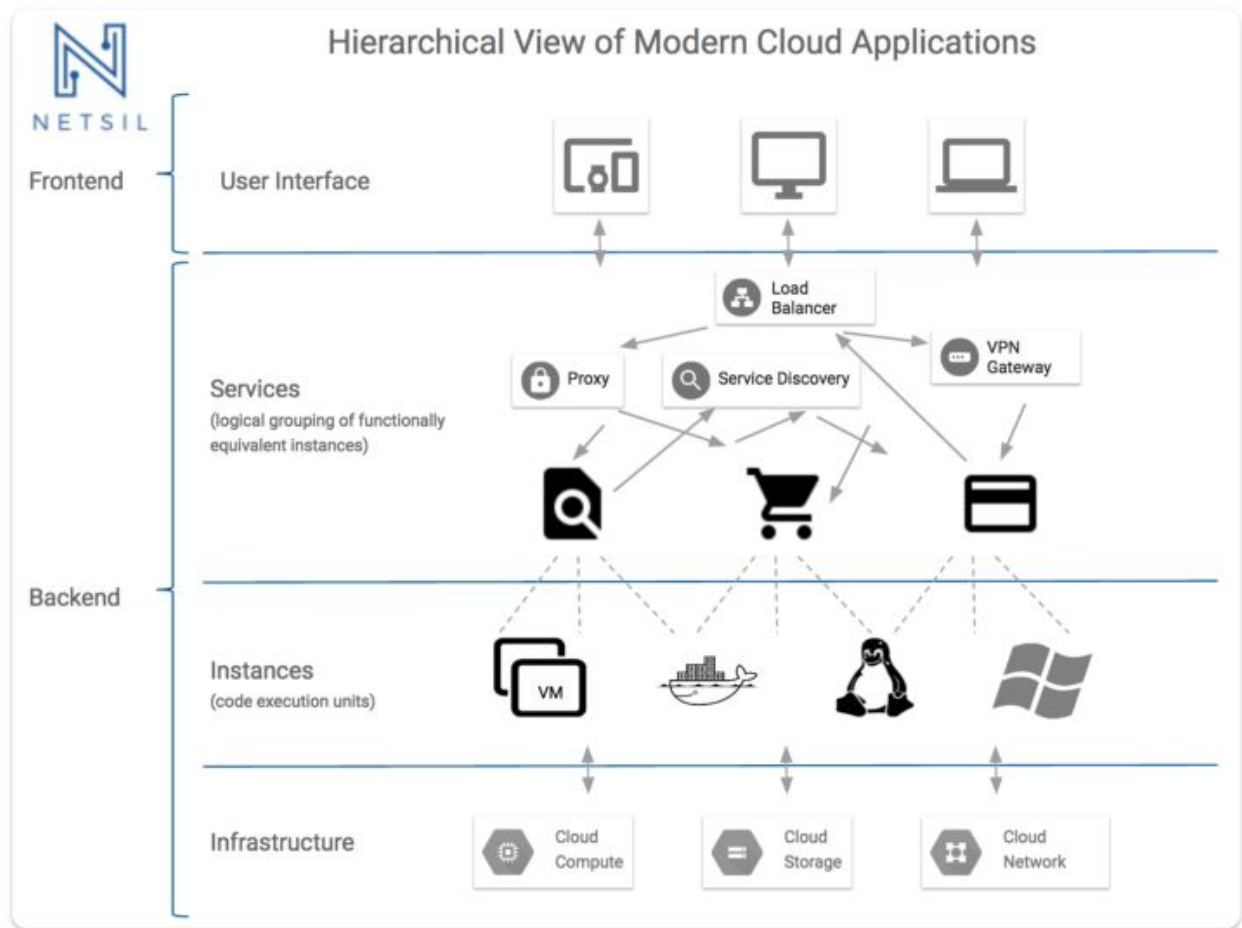
When faced with a complex problem, business leaders, top consultants and bright minds across the globe have often relied on the famous “2x2” framework to establish a simpler perspective. We will follow suit and develop a 2x2 categorization of the monitoring landscape. Specifically, we will contrast and categorize along:

1. Service-level vs Instance-level monitoring
2. Black-box vs White-box monitoring

Before we discuss these categories, it is worth establishing a hierarchical view of modern applications (as shown in figure 2).

1. At the top level, users interact with any business application via frontend interfaces such as mobile apps, browsers, TV apps, consumer devices such as Alexa, etc.
2. User interactions are communicated to the application **backend** (i.e. server side) via APIs. Each user request is realized through complex coordination among many services. **Services** are logical grouping of functionally equivalent instances. For e.g. multiple instances could be providing a REST service for the item list in a shopping cart. Since all these instances are serving the same REST endpoint (or URI), they can be grouped to form a “list shopping cart items” service. In modern applications, services are the natural grouping of functionality within an application; hence, services are a critical abstraction level in the application hierarchy.
3. Services are realized by multiple **instances** of code running inside VMs, containers or bare metal operating systems.
4. The instances run on the **infrastructure**, which forms the lowest layer.

In this white paper, we will focus on the backend monitoring. As a result we are skipping techniques such as synthetic transactions, or methods used for monitoring user interfaces such as mobile apps.



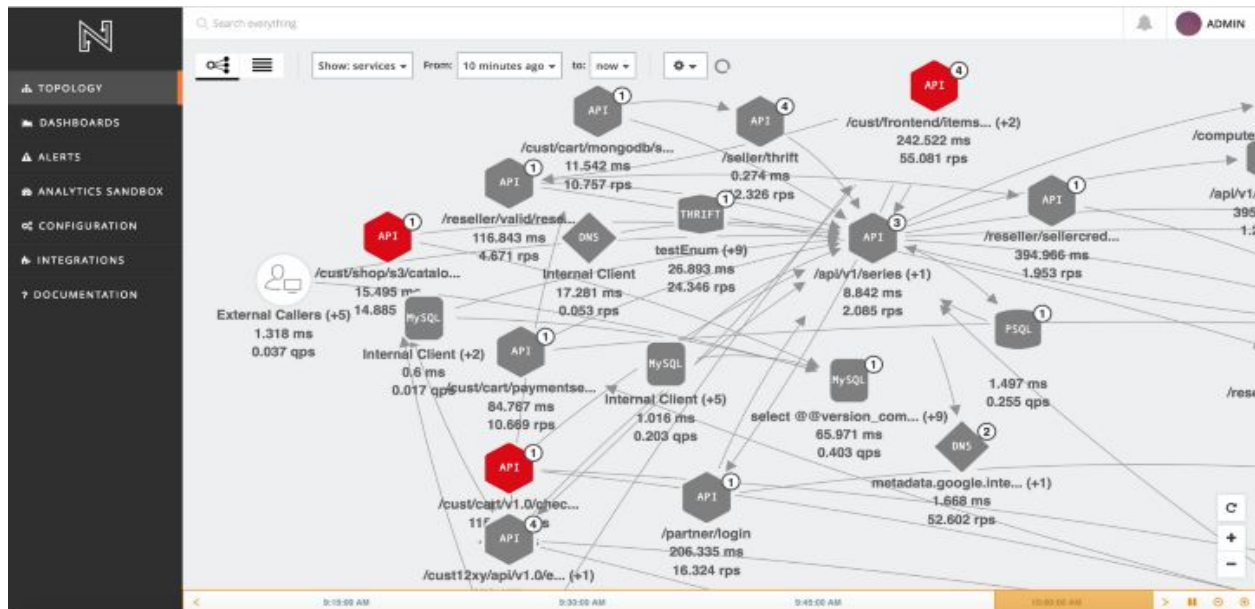
*Hierarchical View of Modern Applications*

## Service-level Monitoring

The category is also referred to as Objective-oriented monitoring

For backend monitoring, the service-level monitoring approach starts by auto-discovery of services i.e grouping of functionally equivalent instances. The dependency among services is captured using techniques such as network communication analysis or distributed tracing. Once services and their dependencies are established, service-level monitoring is able to establish a topology map of the application.

In an application topology map, each node represents the service and each edge represents the dependency between the services as determined by network communication. With this topology map, operations teams (Site Reliability Engineers (SREs), DevOps engineers, DBAs, etc.) can easily define Service-level Objectives (SLOs) for golden signals such as service latency, throughput, error rates and saturation. The importance of service-level monitoring has been highlighted by many famous SRE practitioners including Brendan Gregg at Netflix, Kyle Brandt at StackExchange and in the Google SRE bible.



*An Example of Application Topology Map in Netsil Application Operations Center (AOC)*

---

## Instance-level Monitoring

The category is also referred to as Diagnostic-oriented monitoring

This is the traditional monitoring which involves gathering logs and metrics from individual instances (application code, VMs, containers, etc.) and underlying hardware components. While service-level monitoring tracks SLOs, the instance-level monitoring is needed particularly for diagnostics. After all, services are an abstract grouping and for diagnosing the health of services we need metrics for underlying instances and their runtime environments.

The primary evolution in the instance-level monitoring has been the application of time-series database and stream processing techniques to handle the exponential growth in metrics, improve the metrics resolution to sub-second collection and deliver powerful real-time analytics engine.

## White-box Monitoring

Another categorization of monitoring approaches involves whether the monitoring technique requires changes to the application code or not. The techniques that require code changes, either at implementation or at run time, can be categorized as white-box monitoring. Some popular techniques in this category include:

- Log based monitoring techniques need logs to be implemented in the code.
- Application Performance Management (APM) techniques such as byte-code instrumentation require run-time (or load-time) changes.
- Use of frameworks such as statsd to instrument code and send custom metrics to an analytics engine.
- Finally, most of the distributed tracing techniques that require wrapping up the code or API calls with transaction identifying metadata. Google Dapper remains the seminal work for further reading on distributed tracing.

---

## Black-box Monitoring

As you might have guessed, these set of techniques do not involve changes to the code. That means neither code change at implementation nor at the run time. The options to gain insight into an application without any code change include:

- **Leveraging Operating System** support for gaining insights into application execution. In addition to basic metrics related to cpu, memory, disk, network, etc., most modern operating systems provide advanced capabilities to gain application run time insights. At broad stroke the idea is similar to byte-code instrumentation, the difference is that the operating system is providing the probes and hooks to gain insights into the application. For linux, the upcoming kernel version greatly expands on the capabilities to provide visibility into the application. We would highly recommend consulting Brendan Gregg's comprehensive analysis of linux kernel advancements in this context.
- **Leveraging network communications** to gain insights into services and applications. With the rise of service-oriented and microservices architectures, the complexity has shifted from the code to the network. Comprehensive insight into the health and performance of applications can be obtained by looking at network communication metrics such as throughput, latency, packet retransmissions, byte size of request/response, etc. This approach doesn't require any code change and relies on well established remote packet capture techniques supported in both windows and linux operating systems. Netsil is the pioneer in this approach and more details are captured in [this blog](#).



# Application Monitoring Categorization Framework




## CATEGORIZATION OF APPLICATION MONITORING TECHNIQUES

	<b>INSTANCE-LEVEL</b> (DIAGNOSTIC-ORIENTED MONITORING)	<b>SERVICE-LEVEL</b> (OBJECTIVE-ORIENTED MONITORING)
<b>BLACK-BOX</b> (NO CODE CHANGE REQUIRED)	<ul style="list-style-type: none"> <li>Using collectors such as Datadog agent, Nagios, Telegraf, etc.</li> <li>Using operating system based insights such as Berkeley Packet Filter (BPF)</li> </ul> <p>Datadog, Nagios, Sysdig</p>	<ul style="list-style-type: none"> <li>Network packet capture leveraging Pcap</li> </ul> <p>Netsil</p>
<b>WHITE-BOX</b> (CODE CHANGE REQUIRED EITHER AT IMPLEMENTATION OR RUN TIME)	<ul style="list-style-type: none"> <li>Code instrumentation such as Java byte-code instrumentation</li> <li>Custom metrics using Statsd</li> <li>Log analysis</li> </ul> <p>New Relic, AppDynamics, Instana, Splunk, Sumo Logic</p>	<ul style="list-style-type: none"> <li>Distributed tracing frameworks such as Google Dapper, Twitter Zipkin</li> <li>Tracing using specialized proxies such as Linkerd</li> </ul> <p>Lightstep, Buoyant</p>

*Categorization of Application Monitoring Techniques*

We promised to boil all this complexity into the omnipotent 2x2 framework. Figure 4, captures the monitoring techniques categorized into: black-box vs white-box and service-level vs instance-level. Few important points to note:

- Black-box monitoring techniques can also ingest custom metrics from the application (*for e.g., using statsd*).
- We have put APM or more specifically byte-code instrumentation based technique in the White-box + Instance-level category, since they require changes to application run time and are gathering insights on individual code instances.
- Distributed tracing techniques, most of which rely on some code change, provide insights across multiple services. They don't necessarily create any grouping of instances into services but still provide top-level insights into the flow of API requests across instances. Hence, we have categorized them into White-box + Service-level monitoring category.



Finally, a note on the companies mentioned in the framework. Companies and their products leverage a combination of monitoring techniques. So even though companies are mentioned as an example in one category, their offerings might deliver value from multiple techniques. Netsil, for example, uses a combination of network packet capture to auto-discover services and uses other agents to collect instance level metrics such as docker container statistics. So Netsil is able to provide both service-level as well as instance-level insights.

With the simple 2x2 categorization, let's explore how it can be used by *dev* and *ops* personas to evaluate the many choices in monitoring techniques.

## Mapping Personas to Monitoring Techniques

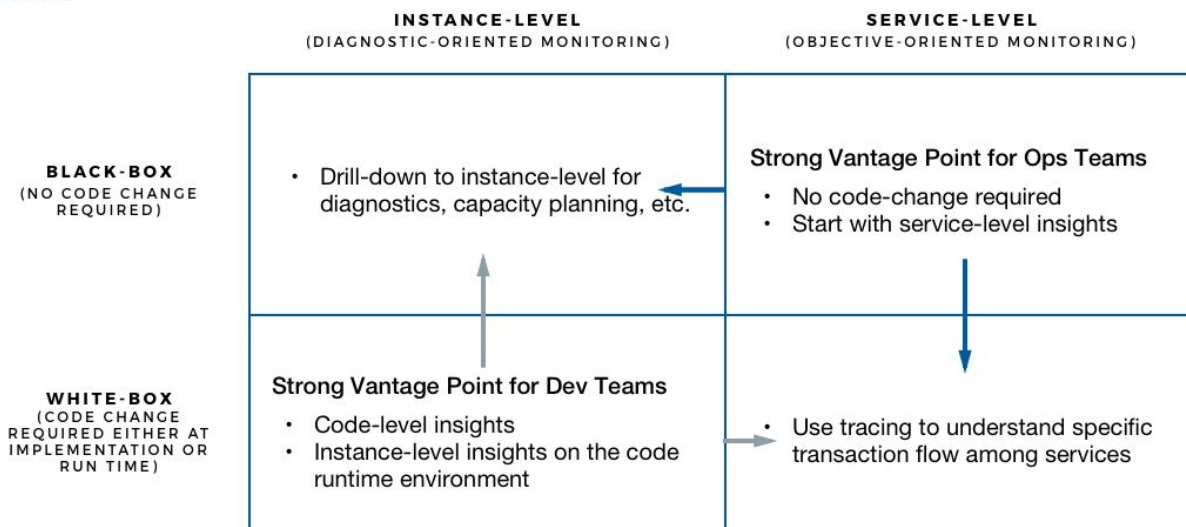
There are two simple factors that can help map the above framework to personas and select the right set of monitoring products.

1. **Level of familiarity with the application code:** Dev teams implementing the services are intimately familiar with the code. So for the dev teams, white-box techniques such as byte-code instrumentation and logs will provide meaningful insight. Whereas for the ops teams (SREs, Devops engineers, DBAs, etc.), who primarily inherit the service in production, the code is a black-box. Code-level insights such as function call stack or intricate logs have limited value for ops teams, who are unfamiliar with the code logic. For ops teams, black-box monitoring techniques which don't require any code change or code-level understanding are ideal.
2. **Number of services that need to be monitored:** Again if you are dev team responsible for handful of services, then instance-level and white-box metrics will be sufficient. Whereas for ops teams responsible for the health of several dozens of services, service-level and black-box monitoring will be more productive. Monitoring each and every individual instances will simply swamp ops teams with thousand of meaningless alerts resulting in massive [alert fatigue](#). For ops teams, service-level insights into latency, throughput, error rates, etc. are more meaningful.

In short, service-level + black-box monitoring techniques are ideal for ops teams focused on health and performance of many production services. Whereas instance-level + white-box techniques are more meaningful for dev teams interested in performance and health of individual services at the code-level. Both the personas can drill-down to instance-level metrics for diagnostics, capacity planning, etc. And both dev and ops teams can use white-box distributed tracing mechanisms for insights into individual transactions flowing through the application.



#### MAPPING PERSONAS (DEV, OPS) TO MONITORING TECHNIQUES



*Mapping Personas to Monitoring Techniques*

---

## Conclusion

Our goal with this whitepaper was to provide a simple framework to help navigate and choose the right monitoring techniques. At Netsil, we have pioneered a powerful, elegant approach that is a combination of:

- Black-box monitoring i.e doesn't require any code change, and
- Service-level monitoring i.e automatically discovers services, allows defining service-level objectives and then enables drill-down into instance-level metrics

We firmly believe that the Netsil service-level monitoring approach is the fundamental necessity for operations teams managing modern cloud applications. We would encourage you to download and get started with the Netsil Application Operations Center (AOC) and engage with us for improving the reliability and performance of your business critical applications.