DZone

THE DZONE GUIDE TO

Cloud

Native Development & Deployment

VOLUME IV

BROUGHT TO YOU IN PARTNERSHIP WITH

ca technologies

exoscale

IBM

DEAR READER,

What would you do if you won the lottery?

I'm sure you've been asked this question before, or at least considered it. If you're like me, your immediate answer would be rooted in the constraint-driven reality you have been living in your whole life. You would do the things you have always done, just do more of them: travel more, buy *more* things, take your friends and family out more.

But at the heart of the question is a notion that makes it truly exciting: you will no longer be bound by one of the biggest constraints that has shaped your reality, a notion that begs a reevaluation of your assumptions.

In the software world, cloud-based development is presenting a similar paradigm. The promise of the cloud is the elimination of the underlying constraints that have existed since the beginning of software development: no longer will you have to be concerned with provisioning hardware or limiting your application by available, local computing resources.

So, what does this mean for you, the modern software developer?

At the core, this new reality is an invitation to open your imagination and strive for technical elegance rather than constraint-driven design. You must imagine and build solutions that don't just do *more* of the same, but rather begin with new assumptions, where resources are practically limitless, and you are only limited by your own imagination and capabilities.

But, just like if you won the lottery, cloud-based development also brings with it an entirely new set of challenges.

Cloud providers, like forgotten or unknown relatives, will court you and vie for your attention and money. You will have to be leery of coupling your application too closely to an ecosystem of managed services to avoid the dreaded "vendor lock-in." That doesn't go to say a lottery winner can't find their soulmate, or that you cannot find the perfect service provider (just be sure to sign a pre-nup).

With your data changing hands more often, security will have to become one of your primary concerns. You will need to learn what to look for when deciding who to trust.

Most importantly, you will be faced with constant self-reassessment, determining where the limitations of your abilities to architect solutions, not the availability of resources, are creating a bottleneck.

So, whether you are still simply daydreaming of a world free of the constraints you have grown accustomed to, or you have already "won the lottery" and are building in the cloud, we hope this Guide will help to steer you through this rapidly shifting set of challenges, while opening your mind to new possibilities.

Good luck!

BY KELLET ATKINSON

DIRECTOR OF MARKETING, DZONE

RESEARCH@DZONE.COM

TABLE OF CONTENTS

PRODUCTION

CHRIS SMITH
DIRECTOR OF PRODUCTION

ANDRE POWELL
SR. PRODUCTION COORDINATOR

G. RYAN SPAIN
PRODUCTION PUBLICATIONS EDITOR

ASHLEY SLATE
DESIGN DIRECTOR

MARKETING

KELLET ATKINSON
DIRECTOR OF MARKETING

LAUREN CURATOLA
MARKETING SPECIALIST

KRISTEN PAGÀN
MARKETING SPECIALIST

NATALIE IANNELLO
MARKETING SPECIALIST

MIRANDA CASEY
MARKETING SPECIALIST

EDITORIAL

CAITLIN CANDELMO
DIRECTOR OF CONTENT + COMMUNITY

MATT WERNER
CONTENT + COMMUNITY MANAGER

MICHAEL THARRINGTON
CONTENT + COMMUNITY MANAGER

MIKE GATES
SR. CONTENT COORDINATOR

SARAH DAVIS
CONTENT COORDINATOR

TOM SMITH
RESEARCH ANALYST

JORDAN BAKER
CONTENT COORDINATOR

BUSINESS

RICK ROSS
CEO

MATT SCHMIDT
PRESIDENT & CTO

JESSE DAVIS
EVP & COO

MATT O'BRIAN
DIRECTOR OF BUSINESS DEVELOPMENT
sales@dzone.com

ALEX CRAFTS
DIRECTOR OF MAJOR ACCOUNTS

JIM HOWARD
SR ACCOUNT EXECUTIVE

JIM DYER
ACCOUNT EXECUTIVE

ANDREW BARKER
ACCOUNT EXECUTIVE

CHRIS BRUMFIELD
SR. ACCOUNT MANAGER

ANA JONES
ACCOUNT MANAGER

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

SPECIAL THANKS
to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this guide a great success.

Executive Summary

BY MATT WERNER
CONTENT AND COMMUNITY MANAGER, DZONE

Cloud computing is slowly becoming seen as the "new normal." Most new applications that are released today are following the SaaS subscription model, and most developers have experience using platforms or infrastructures-as-a-service. That's not to say the industry isn't continuing to evolve, offering new benefits and new problems. Serverless computing is starting to emerge as a way for teams to worry less about server management and runtime environments; the importance of security is more evident with every data breach that makes the news; and deploying and managing software containers in production is still one of the bigger challenges for today's developers. We've assembled this research guide to help navigate these new developments and get a snapshot of where the industry is today. Seven industry experts have shared their thoughts on best practices, serverless computing, security, and more. We've reviewed the modern history of cloud computing from x86 Virtual Machines to Kubernetes and Docker Swarm, and we've analyzed our annual survey of almost 500 developers and IT professionals working with cloud technology.

LOCKING EVERYTHING DOWN

DATA Security was considered a "very important" factor for choosing a cloud provider for 93% of survey respondents, followed by performance (85%), scalability (74%), and support (65%). The most important security features were firewalled servers (71%), data encryption (66%), and authentication options (61%).

IMPLICATIONS An overwhelming majority of IT professionals recognize that security is paramount to the future of their applications and businesses. Respondents are starting to require that their cloud providers have features like firewalled servers, data encryption, and authentication options. The cloud solutions that provide these tools will ultimately be the ones to succeed in the market.

RECOMMENDATIONS Developers have to learn to start "baking in" security into the SDLC. As data breaches continue to make the news and impact everyday lives, it's imperative that in addition to choosing a secure cloud provider, development and security teams learn how to use cloud security tools effectively, as well as protect against threats like cross-site scripting and insider threats on a regular basis. Check Gaurav Purandare's article on building a HIDS in the cloud on page 20 to get an in-depth example of how developers can get started securing their applications.

CLOUD AND DEVOPS GO HAND-IN-HAND

DATA Of respondents who performed deployments to production on a cloud platform, 34% had on-demand deployment capabilities, and were 14% less likely to have an average deployment of once per month. Of cloud-native app deployments, 35% were performed multiple times per day, on-demand.

IMPLICATIONS Cloud technology helps developers improve the deployment pipeline to push changes as soon as possible, which is one of the key principles of DevOps. In addition, 51% of those doing on-demand deployments were using container technology. Similar results in DZone's Guide to Continuous Delivery saw that respondents who used containers were also more likely to have a faster mean time to recovery.

RECOMMENDATIONS While elasticity and speed are key features of adopting cloud technology, failure is always possible. Be sure to read Christian Posta's article on cloud-native applications and preparing for said failure on page 6.

AMAZON, MICROSOFT, AND GOOGLE DOMINATE

DATA 60% of survey respondents use Amazon Web Services (up 4% from 2015), 31% use Microsoft Azure (up 5% from 2016), and 19% use Google Cloud Services.

IMPLICATIONS AWS has taken its early adopter advantage and grown into the dominant vendor in the space due to product improvements and new products, like AWS Lambda, which handles server management and runtime environments. Because of Amazon's widespread use, developers who learn how to use AWS will have transferable skills between several jobs. In addition, the fact that usage of AWS, Azure, and GCS did not increase by a significant amount suggests that the industry is becoming more narrow and less volatile, and that both Azure and GCS are strong competitors.

RECOMMENDATIONS GCS and Azure are still worth considering, even though they're less popular than AWS. For example, GCS users experienced 8% less cloud-caused downtime than AWS users. Serverless computing is starting to take off, and AWS and Google are leading the charge with AWS Lambda and Google Functions, respectively, so it's worth exploring these tools and incorporating them into your architecture. Take a look at what these frameworks bring to the conversation in our infographic on page 16, as well as Imesh Gunaratne's (page 23) and Andreas Wittig's (page 12) articles.

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

498 respondents completed our 2017 Native Cloud Development survey. The demographics of the respondents include:

- 21% of respondents work at organizations with at least 10,000 employees; 17% work at organizations between 1,000 and 10,000; and 26% work at organizations between 100 and 1,000.
- 42% of respondents work at organizations in Europe, and 29% work at organizations in the US.
- Respondents had 15 years of experience as an IT professional on average; 32% had 20 years or more of experience.
- 30% of respondents identify as developers or engineers; 22% as developer team leads; and 20% as software architects.
- 82% of respondents work at organizations that use Java, and 74% work at organizations using JavaScript (36% only using client-side, 5% only using server-side, and 34% using both).

WHAT'S THE DEAL WITH DOCKER?

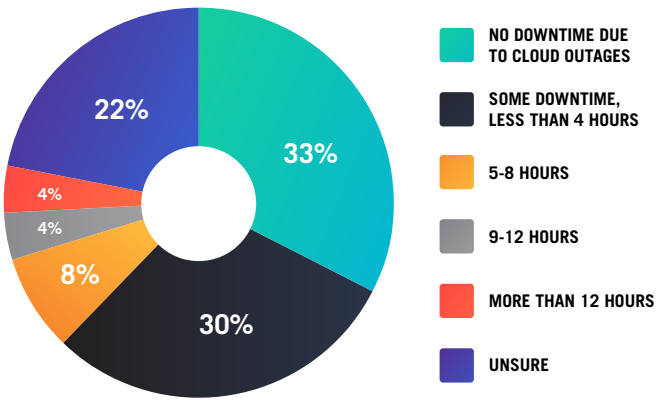
At this point, Docker is nothing new to most enterprise developers. And while Docker certainly isn't the only way to containerize applications, it's hard to think of containers without Docker (and/or that happy whale) coming quickly to mind. Container adoption has had consistently high growth amongst our cloud survey respondents for years now. This year, 39% of respondents said their organization has adopted container technology, compared to 25% in last year's cloud survey and 10% the year before. (It is worth noting that results from the cloud survey results on container adoption differ from two recent Continuous Delivery surveys DZone ran, which had reports of 25% and 30% of respondents' organizations using container technology.) Only 11% of respondents said they weren't even considering containerization.

Discrepancies between the cloud survey and CD survey may be explained in part by the fact that respondents at organizations developing cloud native applications were much more likely to respond that their org used container technologies. 40% said their company develops cloud-native applications vs. 44% who said their company does not (16% were not sure). Those who don't work at companies developing cloud-native apps were 24% likely to say their company has adopted containers, while 58% of cloud-native company respondents also answered that their company uses container technology. Furthermore, there was a correlation between container usage and deployment frequency, with 51% of respondents whose companies have daily or on-demand deploys had adopted containers, compared to 39% who had weekly deploys, and 23% and 25%, respectively, for companies who had monthly and quartly deploys.

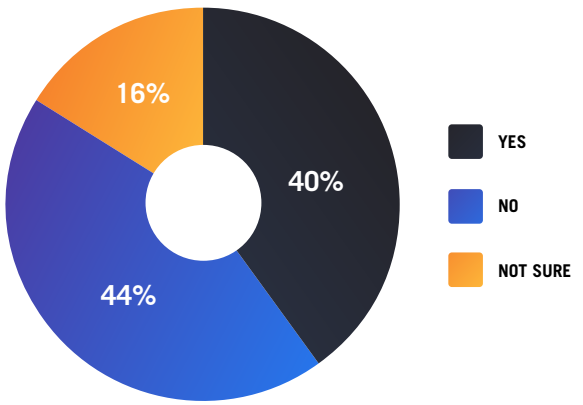
CLOUD GIANTS

The use of different cloud service providers, unlike container usage, has been incredibly stagnant for the last few years. Between our 2015 cloud survey, our 2016 cloud survey, and this year's survey, there was almost no statistically

HOW MUCH DOWNTIME DID CLOUD OUTAGES CAUSE YOUR COMPANY LAST YEAR?



DOES YOUR COMPANY DEVELOP CLOUD-NATIVE APPLICATIONS?



significant change in the use of cloud service providers reported on. Usage of Amazon Web Services, the most popular cloud service provider all three years, grew 4% between 2015 (56%) and 2017 (60%). Microsoft Azure had slightly more growth, with usage up 5% from 2016 (31% compared to 26%) and up 7% from 2015 (24%). So AWS is still almost twice as likely to be used as Microsoft Azure, and three times as likely as Google Cloud Services (19% in 2017).

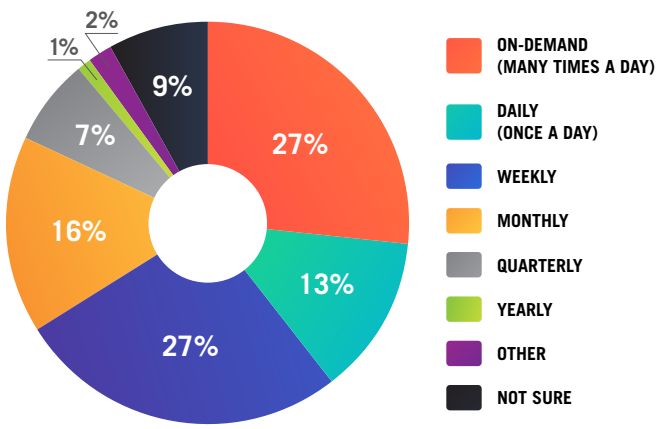
While Amazon remains on top, the static usage of Azure and GCS indicate that they won't be going away anytime soon, and respondents using different service providers had different experiences with cloud development on average. For example, users of Google Cloud services were 8% more likely to have low cloud-caused downtimes (0-4 hours over the span of a year) compared to Amazon and 11% more likely compared to Azure. On the otherhand, Google Cloud Services users were also 8% more likely to claim they experience cloud platform issues like integration, security, and application performance.

DEPLOY AHOY

Application deployment frequency varied quite a bit amongst our respondents. 27% of respondents said their organization has on-demand deploys, while 17% said they only deploy about once a month. As mentioned earlier, container usage correlated with faster deploy times. Another factor that seems to have an impact is actually performing prod/deploy on a cloud platform. Respondents who said they perform production/deployment on a cloud platform were 11% more likely to have on-demand deploys (34%) than those who don't (23%), and were 14% less likely to have an average deployment frequency of monthly or greater (21% vs. 35%).

Additionally, respondents at companies developing cloud-native applications were also more likely to deploy more frequently. 35% of cloud-native application deployments were estimated to be on-demand (multiple times per day), compared to 21% of non-cloud-native apps. And 53% of on-demand deployments were done by cloud-native app companies, as opposed to 34% done by companies not developing cloud-native applications.

ON AVERAGE, HOW OFTEN DOES YOUR ORGANIZATION DEPLOY CODE?

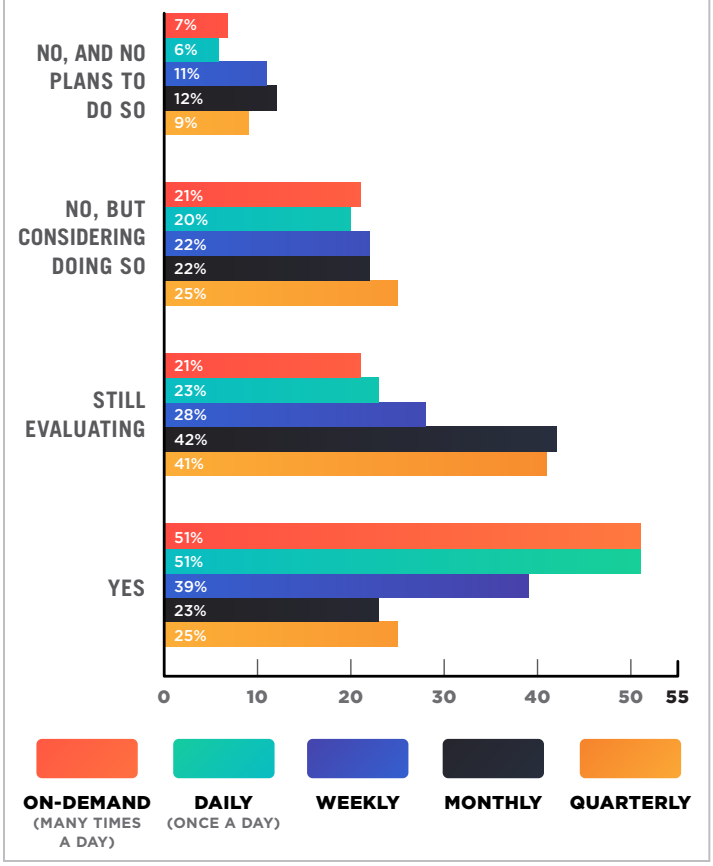


THE IMPORTANT THINGS

Virtual Machines form the foundation of many organizations' infrastructures, cloud native or not. 86% of respondents said their company uses VMs, up slightly from last year's 82%. The focuses for optimizing the configuration of VM instances has remained about the same for the past year, with CPU and memory being most commonly optimized (63% and 60% of respondents said their orgs optimize for these, respectively), followed up by storage volume and storage throughput (39% and 31%, respectively), and lastly GPU (8%). None of these VM optimizations had a difference from last year's responses outside of the margin of error.

When selecting a cloud service provider, there were also some factors respondents held much more important than others. Overall, security was the #1 influence on cloud service provider selection, with 93% of respondents rating it as "very important." Next were performance (85%), scalability (74%), support (65%), and price (61%). Specifically regarding security factors in a cloud provider, respondents found firewalled servers to be most crucial (71% said "very important"), followed by data encryption (66%), and authentication options (63%).

CONTAINER ADOPTION VS. AVERAGE DEPLOYMENT FREQUENCY



Cloud Native Applications and the CAP Theorem

QUICK VIEW

01 A practical understanding of cloud-native applications.

02 How the CAP theorem plays into cloud applications.

03 Thinking about failure domains is critical to developing cloud applications.

BY **CHRISTIAN POSTA**

PRINCIPAL ARCHITECT AT **RED HAT**

Building applications for enterprises has come a long way. To simplify a bit, we've gone from simply automating paper processes to using it as an alternative channel for our businesses to becoming the business itself. Parallel to this we've seen the evolution of Moore's law from cheaper, faster individual hardware to fast, cheap, piles of hardware on demand. If our applications "are the business," and we have access to fleets of cheap machines and infrastructure, what does this mean for the applications and developers in this new era? It means we have to innovate by making changes quickly. We have to build resilient systems. We need to scale to deal with challenges in the world of IoT, big data, etc. Building systems and applications that fit this "cloud era" are a bit different than how we've built them in the past.

In the past, we developers had a lot of safety guarantees that made our job a lot easier. Everything ran mostly co-located. When things failed, we knew where to look and could reason about stack traces from our co-located components. We didn't have to reason much about concurrency or failure conditions with our data because the database solved a lot of that for us. When we called application or infrastructure services we did so with confidence because we were told those services were highly available (whatever that really meant). When we made changes to the data owned by our application, those changes were instantly visible to other components in our application. We could make changes to many components with a single deployment since we owned all of the application. And there are many other assumptions we've lived with, many of which will not hold up in the architectures designed for the cloud era.

When we think about designing applications in a cloud-native world, we focus on three main things:

1. Speed of change: Can we make changes to the overall system without impacting other applications or services?
2. Elasticity: We should be able to take advantage of additional infrastructure and scale our applications horizontally.
3. Failure: Things can and do fail all the time, whether it's computers, network, storage, or just as likely: our own applications.

For this article, I'll focus mostly on the last bullet: designing for failure.

When we build cloud-native applications we are first and foremost building a distributed system. Communication between services and infrastructure happens over the network. Failure happens in all shapes and sizes. Hardware fails. Applications have bugs. Things can be working perfectly fine, but to an observer across the network they appear to have failed. But services need to communicate and cooperate with each other to make progress. In many cases they share similar concepts and data. For example, for an online bookstore, a book may be represented in different parts of the system differently. We may have a search service, a recommendation service, and a checkout/order service. All of these have a different understanding about what is a book and

how to interpret data related to books. A problem arises when certain details of a book change. All of these services that store information related to a particular book may need to be updated with the new information. How do we deal with this?

Another issue that arises in these distributed, cloud-native architectures is when we have services that rely on the availability of their downstream dependencies. For example, if service A calls service B and C, and cannot make any material progress without getting responses from both B and C, then what do you do in the face of failures? Return an error saying we cannot proceed because other parts of the system are down?

> "If you build the ability to tolerate failure into the system as a first-class citizen, then you have to accept that data across these services may not always be strictly consistent."

As developers in this new cloud era, we have to revisit some of the motivations for why we had it much easier in the past. We'll find that this new cloud-native model blows up many of these previously held assumptions. We must return to the drawing board and be much more steeped in distributed systems practices. For example, although we've probably heard about the CAP theorem, what does it have to do with cloud-native applications? Above, we were discussing interaction patterns between services, especially where distributed data is involved. CAP tells us we can "pick two out of" consistency, availability, and partition tolerance for data consistency. This may be a good "jumping-off" point, but it's not sufficient. Consistency comes in many forms: linearizability, sequential, causal, PRAM, read-your writes, eventual, and many others. The CAP theorem discusses trade-offs in terms of linearizability alone. What about the other consistency models? Are they appropriate? Do they help with some of the issues described above? Doug Terry has a great paper about replicated consistency models in real life that sheds some light on this.

If you build the ability to tolerate failure into the system as a first-class citizen, then you have to accept that data across these services may not always be strictly consistent. We should still be able to move forward, make progress, and provide some level of service to our service consumers and ultimately to our customers. If I make a call to the recommendation service, should it just fail because some of its dependencies are not available? Or does it still have the responsibility to provide some level of "service"? In many ways, services make promises about what they intend to do. There are no absolutes. In the face of failures our services try to do what they can with what they have. Mark Burgess discusses this deeper in his explanation of "promise theory." Services make promises to each other, and in the event they cannot keep their promises, each individual service must do what it can independently to keep its promise. Promise theory makes very explicit the nature of uncertainty and failure in these types of systems. We need to build services from the onset with promises in mind.

If we have failures as a first-class design consideration and we have relaxed consistency models to be able to contend with it, we absolutely can end up in situations where we make a decision that will have to be reverted. For example, if I order a widget on a popular online retailer and they take my order, what happens if they cannot fulfill it? What happens if they run out of stock? Or if it gets lost in delivery? Or they never had any in stock in the first place? In our previous life of comfort and safety we could pessimistically enforce that these situations do not occur. In a complex distributed system, these scenarios can and will crop up. But are they so bad? In the business world, we have ways to work around this. We may just resend the widget if it gets lost in delivery. We may go to our competitor and buy the widget from them and send it along to you in an effort to keep our promise. Have you ever heard of "overbooked" airplanes? If everyone shows up they offer money for people to take a bump. In the real world we deal with these sort of inconsistencies as they pop up and as part of doing business. Why shouldn't we do so in a distributed system environment?

The point of this article was to get you thinking about how distributed environments work in the real world and how we have the tools and practices to accomplish the same in distributed computing. This is not for the faint of heart. The long-standing safety guarantees of our past world should not be given up lightly if not needed. But I suspect in this new cloud world the focus on speed of change, resiliency, and scalability will preclude us from ignoring distributed systems.

CHRISTIAN POSTA (@christianposta) is a Principal Architect at Red Hat and well known for being an author (Microservices for Java Developers, O'Reilly 2016), frequent blogger, speaker, open-source enthusiast, and committer on Apache ActiveMQ, Apache Camel, Fabric8, and others. Christian has spent time at web-scale companies and now helps companies creating and deploying large-scale distributed architectures - many of what are now called microservices based. He enjoys mentoring, training, and leading teams to be successful with distributed systems concepts, microservices, DevOps, and cloud-native application design.

VIRTUALLY EVERYONE IS USING CLOUDS.

ARE YOU GETTING THE MOST OUT OF YOURS?

Accelerate, optimize and secure your clouds.
Become a master of the cloud with CA.

Learn more at **ca.com/cloud**

ca technologies

Living in a modern society almost necessitates the use of certain technologies. Imagine trying to find a decent job—or even line up a date—without a mobile device. Sure, it's possible, but you would be at a significant disadvantage without one.

When technologies reach a certain level of adoption, it becomes very difficult to participate in modern society without using them. We've seen this happen again and again with technologies ranging from electricity to automobiles to internet access.

A very similar thing happens in the business world. When technologies first arrive, early adopters can gain a competitive advantage—though a short-lived one—by successfully leveraging cutting edge technologies. Think of the many examples: Henry Ford's assembly line; Atari's home video game system; the iPhone's touch screen; Tesla's advanced battery technology.

When it comes to cloud computing, we are past the early adopter phase and into the phase where inaction means a significant disadvantage. Today a business not using clouds is like a job-seeker without a cell phone number.

We're moving into an era where the conversations around cloud should be less about the risks of migration, and more about the risks of a poor cloud implementation or strategy.

Just using cloud to solve technology problems (like running an application) is not enough. Instead, organizations should think about the bigger picture: that is how to use clouds to move the business ahead.

An example of how cloud use can lead to business results is DevOps. Study after study has shown the business results that come from a successful DevOps practice: 2.5 times faster revenue growth; and 2 times faster profit growth. Using cloud-based tools, test environments and practices for developing, testing and deploying applications are a great way to accelerate a successful DevOps practice in your organization.

Of course, using clouds successfully is more than just having a business objective. You need insight into how your clouds and applications are performing; you need a security strategy that is baked in from the beginning.

And there are organizations like CA that have the tools and expertise you need to go beyond basic cloud adoption, and into cloud mastery.

WRITTEN BY ARUNA RAVICHANDRAN
VICE PRESIDENT OF DEVOPS AND SOLUTIONS MARKETING, CA TECHNOLOGIES

PARTNER SPOTLIGHT

Cloud Solutions By CA Technologies

ca technologies

Manage, develop & deploy and secure applications across public, private and hybrid clouds.

CATEGORY
Solutions to Manage, Monitor, Accelerate and Secure Clouds

NEW RELEASES
Continuous

OPEN SOURCE
Yes

CASE STUDY

OneNeck IT Solutions LLC offers hybrid IT solutions, including cloud and hosting solutions, for businesses around the country. OneNeck helps customers reduce costs, improve service levels, increase revenues and gain competitive advantage across a broad spectrum of industries, including healthcare, manufacturing, financial services, retail, education and government.

CA Unified Infrastructure Management (UIM) provides OneNeck with a single tool for insights into public, private and hybrid clouds, as well as traditional infrastructure.

With CA UIM, OneNeck was able to consolidate from five monitoring tools down to a single monitoring platform to get end to end insights across Microsoft Azure, private cloud and traditional infrastructure. The results were faster triage times and better end user experience for their customers.

STRENGTHS

- Quickly and securely accelerate application release velocity by leveraging clouds
- Most comprehensive hybrid cloud and IT monitoring solution
- Application management tools that eliminate complexity, reduce costs and increase quality
- Security solutions that protect access to your cloud platforms and applications

NOTABLE CUSTOMERS

- US Cellular
- Lexmark
- Logicalis
- Petrobras
- Vodafone

BLOG blogs.ca.com

TWITTER @CAInc

WEBSITE ca.com/cloud

Managing Service Provider Performance and Risk in the Cloud

BY **NICK KEPHART**
SR. DIRECTOR OF PRODUCT MARKETING AT **THOUSANDEYES**

QUICK VIEW

01 Cloud providers vary greatly in how they manage performance. If you use a co-lo, make sure you have monitoring set up on your infrastructure and ISPs.

02 CDNs may offload traffic for you, but it's up to you to verify caching and the connection to your origin is working as expected.

03 DNS is how users reach your app, so make sure it works. Identify your service providers and monitor them as you would any critical app component.

Deploying applications in the cloud is an exercise in outsourcing responsibilities to service providers in exchange for greater efficiencies and capabilities. But it also means greater risk. The October 2016 DDoS attack on Dyn demonstrated this clearly. Ensuring that your end users have a great application experience means that you need to be more aware and knowledgeable about the service providers that make up key links in your digital supply chain.

We'll cover concrete examples of how you can monitor and better manage risk for five types of critical service providers:

- Cloud or hosting
- Internet service
- Content delivery network
- Managed DNS
- DDoS mitigation

CLOUD AND HOSTING

First and foremost, when you're operating in the cloud, you're operating in an environment run and maintained by a third party. But responsibilities and risks vary based on the type of cloud provider you have.

In the case of IaaS providers, they handle all of the infrastructure management while you provide the code. Examples include cloud providers such as AWS, Google Cloud, Microsoft Azure, Rackspace Cloud, and DigitalOcean. When you're operating in an IaaS environment, your focus is on ensuring that the infrastructure services (servers, containers, load balancers, etc.) provided are performing to expectations. This involves monitoring using the cloud provider's own data, augmented by additional data you collect where you can; for example, using Nagios to monitor servers where you can deploy agent code. In general, you have to trust the monitoring systems of the cloud provider because it is the only source of data for many infrastructure services.

Contrast this with hosting or co-location providers (Equinix, CoreSite, Digital Realty, NTT, Telehouse) that provide network connections, power, and physical space, but are not responsible for servers, storage, or networking infrastructure in your immediate environment. When you're operating in a co-location environment, you instrument the environment just like you would your own data center. You can poll devices, run monitoring appliances, and run agents across a broader set of infrastructure. You'll also need to monitor your network environment more comprehensively, including Internet connectivity via ISPs.

ISPS

Connectivity to your cloud-hosted application takes place through Internet Service Providers, typically chosen for you (in the case of IaaS) or offered as a menu (in the case of co-location). These ISPs are typically international transit providers (Cogent, Level 3, Tata, Telia, Hurricane Electric, NTT America) or regional networks (Comcast Business, Verizon, AT&T, Qwest).

In the case of IaaS, you don't have a direct relationship with the ISPs, so you are at the mercy of your cloud provider to properly manage and maintain connectivity. Many IaaS providers, however, offer direct peering to your corporate network, with the potential for latency and reliability improvements. These direct peering links (AWS Direct

Connect, Azure Express Route) have the potential to lessen your reliance on your cloud provider's ISPs; they won't, however, affect customer-bound traffic.

In the case of co-location environments, you choose your ISP and have a direct (meaning $$$) relationship with them. You are responsible for knowing whether they are meeting their SLAs, providing satisfactory availability and latency, and dealing with any outages. You should ensure that you have network monitoring set up for co-lo environments should your ISP have an outage. This type of monitoring can vary from simple command line tooling, such as ping and traceroute, to more complete packages of active network monitoring.

CDN

For many consumer-facing applications, you'll also use a Content Delivery Network (CDN). CDNs position content nearer to end users, caching portions of your application in the process. They are valuable to offload traffic on an external network, reduce latencies and load times, and provide a level of protection against DDoS attacks on your primary domains. Sometimes your CDN service provider will be the same as your cloud provider (Amazon CloudFront, Google, Microsoft), while other times you will choose one or more CDNs (Akamai, CloudFlare, Edgecast) that complement your cloud provider.

Placing a CDN between you and your users means that you now need to manage the performance and risk. When it comes to CDNs, there are three major factors you should be actively monitoring:

1. Edge performance: Is your CDN serving up content to users from an appropriately close edge location?
2. Caching performance: Is your CDN serving up content from the cache (a cache hit) at the expected rate?
3. Origin connectivity: Is the connectivity from your origin servers to your CDN reliable and performant?

Monitoring CDN performance can get tricky. You'll likely get some basic performance metrics from your service provider. You'll want to validate these metrics and collect additional data points (such as origin connectivity) by actively monitoring your CDN environment from the perspective of the origin as well as the end user, installing probes near your origin and near user locations.

DNS

Another critical component of cloud application delivery is DNS, underlined by several high-profile managed DNS outages in 2016. Most organizations will use an external service, managed DNS, to run authoritative DNS servers on their behalf. DNS services may be provided by your cloud provider (AWS Route 53, Azure DNS, Google Cloud DNS), your CDN provider (Akamai, CloudFlare), or by a separate service provider (UltraDNS, Dyn, Verisign, NS1).

You can choose to use multiple managed DNS providers for a single domain, providing some additional resiliency. This comes at the cost of additional complexity of managing multiple service providers, especially when using advanced geo-based DNS features. You can check your current providers by using the command line utility DIG.

Because DNS forms the initial request from a user to your services, it is essential that one of your DNS providers is available to resolve your domain names. And given the fact that DNS records can be cached, interruptions can both be more muted (because your address record is still in a cache) or more severe (a cached name server record increases the time it takes to failover providers) depending on the circumstances.

Although managed DNS providers run large, highly distributed operations, recent large-scale DDoS attacks have impacted DNS services for many organizations. You'll want to actively monitor your domain resolution to alert you to service provider issues and kick off a failover plan.

DDOS MITIGATION

Applications that are the target of DDoS attacks usually require some sort of cloud-based DDoS mitigation service to improve reachability and dissuade attackers. DDoS mitigation services are offered by CDNs (Akamai's Prolexic, CloudFlare, AWS Shield, Level 3) as well as other infrastructure providers (Verisign, Incapsula). These services sit in between the user and your origin, filtering traffic so that your infrastructure receives as little attack traffic as possible. Given their cost, most organizations opt to use DDoS mitigation services only when an attack is underway, using routing to transfer traffic to the service.

Given the location in the critical path between the user and your application, and the typical high cost of these services, you'll want to understand just how well they mitigate attacks and what performance penalties you incur. In addition to the data from the service provider themselves, you can actively monitor DDoS mitigation services to see how scrubbing centers perform and how much loss and latency increase when these services are turned on.

HOW TO GET STARTED

Understanding these five types of service providers will go a long way toward making your cloud-hosted applications more resilient and performant. In addition to the service providers mentioned above, you'll also want to keep tabs on key APIs used by applications and services in your environment (a topic for another article). Start by categorizing your service providers according to importance and risk, and strategize how you can become more savvy about service provider performance. While you can rely on data from these service providers to give you a baseline view, it is often advisable to 'trust but verify' with additional data sources from other monitoring systems, given the importance and cost of these services.

NICK KEPHART leads Product Marketing at ThousandEyes, which develops Network Intelligence software, where he reports on Internet health and digs into the causes of outages that impact important online services. Prior to ThousandEyes, Nick worked to promote new approaches to cloud application architectures and automation while at cloud management firm RightScale.

Serverless Architectures on AWS

BY **ANDREAS WITTIG**
CLOUD CONSULTANT

QUICK VIEW

01 A serverless platform allows you to run your application without the need to spin up and manage a single (virtual) machine.

02 Being able to focus on software development instead of operating a fleet of servers is the primary driver behind serverless.

03 Dividing your application into small functions is necessary when following a serverless approach.

04 Making use of microservice architectures helps you to achieve that goal. Serverless is made for event-driven architectures.

A serverless platform allows you to run your application—including computing, storing, and networking—without the need for spinning up and managing a single (virtual) machine. This article focuses on serverless architectures on AWS including Lambda, API Gateway, DynamoDB, S3, and more. The architecture patterns can be transferred to other cloud platforms as well.

Being able to focus on software development instead of operating a fleet of servers is the primary driver behind serverless. Alternatively, as Werner Vogels, CTO of Amazon.com, says: "No server is easier to manage than no server." Another important aspect of serverless infrastructure is finely granular billing and extreme scalability.

To give you an impression of typical use cases, here are some serverless applications I have been working on recently:

- Collecting metrics from web application with a REST API in a high load scenario.
- Extracting and storing data from incoming emails containing order and status information.
- Creating a REST API providing a standard CRUD backend.
- Collecting and transforming data within an ETL process in a big data scenario.
- Analyzing log messages from a real-time data stream.
- Building a chatbot that is interacting with multiple APIs in the background.

LIMITATIONS

The ability to execute source code on a highly available and scalable infrastructure is the heart of every serverless platform. AWS Lambda, for example, allows you to run your source code in response to events. AWS provides the computing infrastructure and a runtime environment for JavaScript (Node.js), Java, Python, or .NET Core (C#). A typical serverless application consists of multiple functions. On AWS Lambda, there is an execution timeout of 5 minutes for every function execution. Nevertheless, you can execute the same or different functions in a highly parallel manner.

Dividing your application into small functions is necessary when following a serverless approach. Making use of microservice architectures helps you to achieve that goal. Serverless is made for event-driven architectures. An event is a source for triggering a serverless function and the glue when orchestrating multiple serverless functions forming a system.

Triggering a function with AWS Lambda is possible based on the following events:

- Incoming HTTPS request.
- Changing data stored in a database or object storage.
- Incoming message (e.g. email).

- Publishing a task to a topic or real-time data stream.
- Scheduled event (comparable with a cronjob).

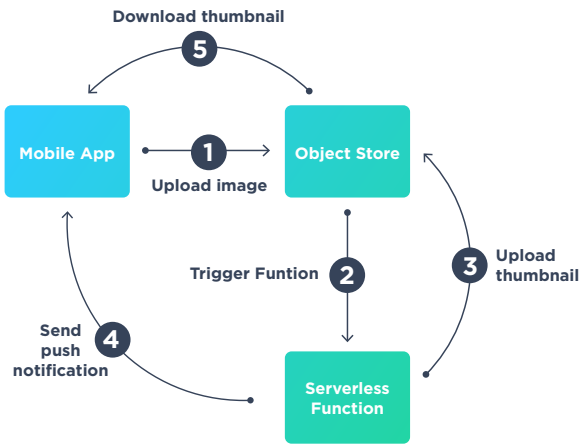
ASYNCHRONOUS MODEL

You can use serverless functions in an asynchronous or synchronous model. In an asynchronous scenario, the caller does not wait until the function returns. Notifications via WebSocket or push notifications to a mobile device can be used to provide feedback to the user.

> "The ability to execute source code on a highly available and scalable infrastructure is the heart of every serverless platform"

The following figure shows an example of an asynchronous serverless architecture. The mobile application is uploading an image to an object store (1). The object storage is creating a change event after the image was uploaded, resulting in an execution of a serverless function (2). The serverless function creates a thumbnail based on the uploaded image. After uploading the thumbnail on the object storage (3) the serverless function sends a push notification to the mobile application (4). The mobile application downloads the thumbnail and updates the user interface accordingly (5).

FIGURE 1: ASYNCHRONOUS SERVERLESS ARCHITECTURE

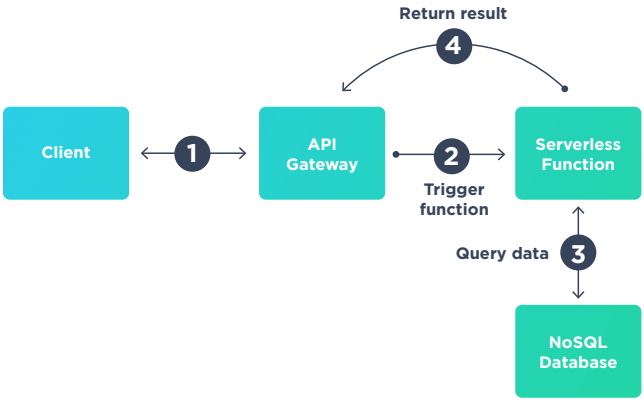


SYNCHRONOUS MODEL

A typical use case for the synchronous model is a REST API. An incoming HTTPS request is triggering a serverless function. The caller waits until the function returns a result.

The following figure illustrates the architecture for a serverless REST API. The client sends an HTTPS request to an API Gateway (1). The API Gateway triggers a serverless function (2). The serverless function reads data from a NoSQL database (3). Afterwards the serverless function returns the result of the database call to the client (4).

FIGURE 2: SYNCHRONOUS SERVERLESS ARCHITECTURE



ARCHITECTURE PATTERN: EVENT SYSTEM

Chaining serverless functions is an anti-pattern. Designing an asynchronous event system is a promising approach instead. An event system receives and processes events by following rules defined inside the system. Processing events happens asynchronously. Asynchronous processing has advantages if you want to build a scalable solution because it frees you from the burden of an immediate response. Instead, the system can queue them and process them as fast as possible.

The following figure shows the architecture of a serverless event system built on AWS IoT and AWS Lambda. An API Gateway receives buy and sell events from a marketplace. A serverless function validates incoming events and redirects them to a topic.

Two rules subscribe to events from the topics.

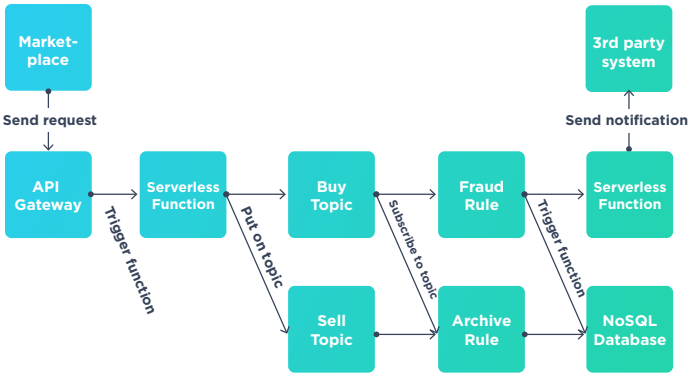
1. The first rule subscribes to buy and sell events. The rule forwards all incoming events to a NoSQL database.
2. The second rule subscribes only to buy events. The rule executes a serverless function for every

incoming buy event. The serverless function checks the buy event for fraud and sends out a notification to another topic whenever fraud is detected.

3. A third rule subscribes to the fraud topic and sends a notification to a third party system for every incoming event.

There is no need for a serverless function to invoke another function. Functions communicate through events distributed via topics.

FIGURE 3: SERVERLESS EVENT SYSTEM



STATELESS

A serverless function is stateless. Persisting state between two executions of the same function is not possible in general. Neither in memory nor on disk. Therefore your serverless function needs to store state externally. On AWS there are the following options to store data: Amazon DynamoDB, a NoSQL database, and Amazon S3, an object store.

> "No server is easier to manage than no server."
>
> **- WERNER VOGELS,** CTO OF AMAZON.COM

Besides that, you can also use old school storage systems like a SQL database (Amazon RDS) or an in-memory database (Amazon ElastiCache). However, beware, some of these old school storage systems do not scale horizontally. So they are a possible bottleneck within your serverless architecture.

> "Persisting state between two executions of the same function is not possible in general."

BUILDING BLOCKS

Serverless is not only about executing source code. Much more important is a platform offering components solving common problems. You can use these building blocks to architect your system and implement your application without reinventing the wheel. Cloud providers offer various managed services. Usually, serverless functions communicate with these managed services via a REST API often wrapped into an SDK. Some examples of building blocks for serverless applications offered by AWS:

- Amazon SES: sending and receiving emails
- Amazon SNS: sending push notifications
- Amazon Lex: building chatbots
- Amazon Polly: turning text into speech
- Amazon Rekognition: analyzing images
- Amazon Machine Learning: using machine learning
- Amazon Cognito: authenticating and authorizing users

SUMMARY

Serverless architectures offer new possibilities for building cloud-native applications. The main benefits of serverless architectures: minimal effort to deploy and operate applications, highly scalable and fault tolerant environments, and the possibility to combine building blocks to solve common problems instead of reinventing the wheel. Event-driven architectures are a perfect fit for serverless functions, as they allow you to couple different components loosely. Splitting tasks into small chunks of work is essential due to the execution duration of a few minutes.

ANDREAS WITTIG is a Cloud Specialist focusing on AWS and DevOps. He is Author of Amazon Web Services in Action (Manning). He helps his clients to gain value from Amazon Web Services. As a software engineer he develops cloud-native applications. He migrated the complete IT infrastructure of the first bank in Germany to AWS.

Diving Deeper

INTO CLOUD

TOP #CLOUD TWITTER FEEDS

To follow right away

 @BButlerNWW

 @Kevin_Jackson

 @GregorPetri

 @DavidLinthicum

 @RandyBias

 @JeffBarr

 @bgracely

 @jamesurquhart

 @Archimedius

 @Werner

TOP CLOUD REFCARDZ

Cloud Foundry

dzone.com/refcardz/cloud-foundry

Covers technologies supported by Cloud Foundry, finding a hosting provider, writing apps for and deploying to Cloud Foundry, scaling and managing apps in the cloud, and more.

Kubernetes

dzone.com/refcardz/kubernetes-essentials

Containers weighing you down? Kubernetes can scale them. In order to run and maintain successful containerized applications, organization is key. Kubernetes is a powerful system that provides a method for managing Docker and Rocket containers across multiple hosts. This Refcard includes all you need to know about Kubernetes including how to begin using it, how to successfully build your first pod and scale intelligently, and more.

Docker Monitoring

dzone.com/refcardz/intro-to-docker-monitoring

Docker has rapidly evolved into a production-ready infrastructure platform, promising flexible and scalable delivery of software to end users. This Refcard looks into the challenges that containers (or black boxes) present in DevOps, explores architectural models for monitoring containers, and investigates pros and cons of Docker monitoring and troubleshooting options. It also covers a complex, real-world example with Sysdig Cloud to understand what to monitor and how.

CLOUD ZONES

Learn more & engage your peers in our cloud-related topic portals

Cloud

dzone.com/cloud

The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. The Cloud Zone focuses on PaaS, infrastructures, security, scalability, and hosting servers.

Internet of Things

dzone.com/iot

The Internet of Things (IoT) Zone features all aspects of this multifaceted technology movement. Here you'll find information related to IoT, including Machine to Machine (M2M), real-time data, fog computing, haptics, open distributed computing, and other hot topics. The IoT Zone goes beyond home automation to include wearables, business- oriented technology, and more.

Security

dzone.com/security

The Security Zone covers topics related to securing applications and the machines that run them. The goal of the Zone is to help prepare the people who make and support those applications stay up to date on industry best practices, remain aware of new and omnipresent threats, and help them to think "security-first."

CLOUD PODCASTS

The Cloud Casts thecloudcast.net

GCP Podcast gcppodcast.com

The Doppler Cloud Podcast cloudtp.com/doppler/podcasts

CLOUD WEBSITES

Netflix Tech Blog techblog.netflix.com

CloudTweaks cloudtweaks.com

CloudAve cloudave.com

the rise of

CLOUD ARCHITECTURE

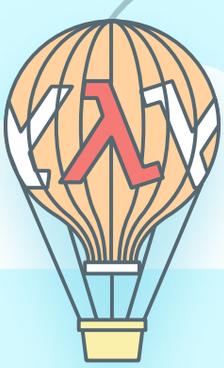
The birth of cloud computing can be traced back to the first hardware virtualization tool in 1967, IBM's CP-40, and when users at large companies once sent jobs to be run on a separate mainframe that their local systems couldn't handle. It's incredible how fast cloud computing rose from a new way of running companies and applications, to becoming a marketing buzzword to "cloudwash" new SaaS applications, to finally becoming the "new normal." However, there's still plenty of work to be done to achieve all the potential of cloud services. Take a brief trip to see how far cloud has come since the 90s, and that all the hype isn't just hot air.

THE
Future
Streamlining containers in production and focusing more on serverless frameworks

39% of DZone users work in organizations that have adopted containers

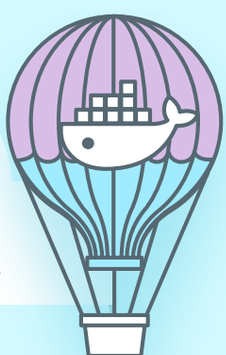
2017
AWS is the dominant cloud provider today

2016
Kubernetes and Mesos are released to manage containers

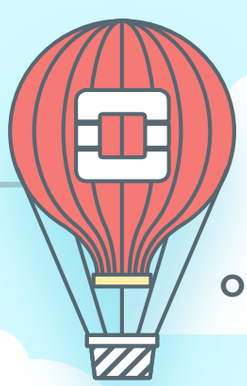


2014
AWS Lambda: the first major serverless offering

2013
Docker containers are released, and Adrian Cockcroft popularizes microservices



92% find security very important



2010
Azure and OpenStack released

40% use some sort of serverless framework

2006
Amazon Launches AWS EC2

60% of DZone users use AWS

1990s
VMware and x86 Hardware Virtualization

86% of DZone users incorporate virtual machines in their infrastructures

IBM

IBM **API** Connect

API Connect integrates IBM StrongLoop and IBM API Management with a built-in gateway, allowing you to create, run, manage, and secure APIs and Microservices.

Unparalleled, integrated user experience.

ibm.biz/apiconnect

Letting Go of your Server with OpenWhisk

As a jaded, old (wait, no "experienced") developer, I've looked at "serverless" with a bit of scorn. Of course there's still servers involved, I told anyone who listened. The name just doesn't make sense! But after giving it a shot with OpenWhisk, I have to say I've become a convert and have wholeheartedly let go of my server.

Given that "serverless" still involves a server, what does it actually mean? When setting up an application on the Internet, developers have to worry both about their particular business logic as well as how they handle the actual traffic/access to their code. So for example, my business logic may be just "Return the time", and that code is relatively simple (ok, very simple), but I also have to setup a server, listen on a port, wait for a particular kind of request, then call my business logic, and finally, return the result in the proper format.

With OpenWhisk, and serverless, my focus is on just the business logic. I can let OpenWhisk, running on IBM Bluemix, handle all the crud around handling requests and just focus on the code itself. For example, the age-old "Hello World"

```
function main(args) {
  if(!args.name) args.name = "Anonymous";
  return { result: "Hello, "+args.name };
}
```

Once this is pushed to OpenWhisk (via a handy-dandy CLI or via the online editor), I'm done. I can access my code via multiple methods including REST if I set it up.

OpenWhisk includes support for multiple languages, support for sequences of events, rules that fire based on external triggers, and more. To learn what you can do with it, see the docs at developer.ibm.com/openwhisk and check out the open source project at openwhisk.org.

WRITTEN BY RAYMOND CAMDEN

NODE.JS DEVELOPER EVANGELIST, **IBM**

PARTNER SPOTLIGHT

API Connect By Strongloop and IBM

IBM API Connect is a complete solution that addresses all aspects of the API lifecycle - Create Run, Manage, Secure - for both on-premises and cloud environments.

CATEGORY
API Management

NEW RELEASES
TBA

OPEN SOURCE
No

API LIFECYCLE

IBM API Connect offers features to manage the API lifecycle, including:

CREATE—create high-quality, scalable and secure APIs for application servers, databases, enterprise service buses (ESB) and mainframes in minutes.

RUN—take advantage of integrated tooling to build, debug and deploy APIs and microservices using the Node.js or Java.

MANAGE—create and manage portals that allow developers to quickly discover and consume APIs and securely access enterprise data, and monitor APIs to improve performance.

SECURE—Administrators can manage security and governance over APIs and the microservices. IT can set and enforce API policies to secure back-end information assets and comply with governance and regulatory mandates.

STRENGTHS

- Simplify discovery of enterprise systems of record for automated API creation
- Provide self-service access for internal and third-party developers through a market-leading gateway
- Ensure security and governance across the API lifecycle
- Unify management of Node.js and Java microservice applications
- Increase flexibility with hybrid cloud deployment

FEATURES

- Unified Console
- Quickly run APIs and microservices
- Manage API's with ease
- Readily secure APIs and microservices
- Create API's in minutes

BLOG developer.ibm.com/answers/topics/apim

TWITTER @ibmapiconnect

WEBSITE ibm.com/software/products/en/api-connect

How to Implement HIDS in the Cloud

BY **GAURAV PURANDARE**
PRINCIPAL ARCHITECT AT **BUILT.IO**

QUICK VIEW

01 The proposed architecture for OSSEC leverages the utilities that many enterprises either already have in use or can easily be deployed today.

02 The proposed implementation focuses on distributed architecture. It is essential to consider this point when the component will be an integral part of every application server.

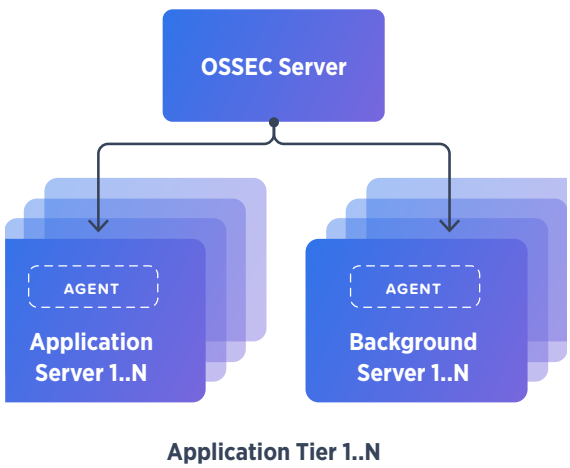
03 By deploying this solution to a large number of servers, OSSEC admins can receive important event notifications in an efficient, targeted manner without being inundated with irrelevant messages.

Ensuring system security is as important as ensuring overall application security. A Host-based Intrusion Detection Systems (HIDS) provides the ability to identify, detect, and notify any unanticipated system changes that might impact the security of the system. HIDS is a powerful tool to maintain security standards implemented across IT systems. The Open Source HIDS SECurity (OSSEC) tool is one of the more popular HIDS options around.

OSSEC ARCHITECTURE

The OSSEC tool can be implemented in multiple ways and largely depends on your infrastructure: whether you have a fixed number of servers or you have infrastructure that is frequently scaled up or down.

TRADITIONAL SERVER-AGENT MODEL



The traditional server-agent model can be followed to monitor the servers with a lightweight agent installed on them, where all agents report to the central OSSEC manager (server) to analyze the events and to generate event notifications as appropriate. This model works quite well for small and large fixed setups. However, there are trade-offs:

- Additional server management overhead. You need to manage the OSSEC server itself, along with its agents.
- Adding or removing agent overhead. If you have a dynamic environment that involves frequent scaling of servers, then a custom solution is needed to automatically register and de-register the agents in the OSSEC server when scaling occurs.
- Limited high availability of OSSEC server. There is an open feature request for making the OSSEC server highly available. Until this has been addressed, your best option is to keep a standby server with the configuration and data volumes replicated in realtime.

LOCAL MODEL

The OSSEC can also be installed in a local model, where each server monitors itself. The local OSSEC installation eliminates maintenance overhead for the OSSEC server. In addition, each server reports incidents and unauthorized events.

Following you will find the steps required to install and incorporate the local model OSSEC into a dynamic infrastructure.

OSSEC INSTALLATION

First, you need to install the required packages:

```
apt-get update
apt-get install build-essential inotify-tools
```

The `build-essential` package will install the required packages for the compilation steps involved in OSSEC installation. The `inotify-tools` package is used in real-time notifications.

Download and extract the OSSEC source code:

```
wget -O ossec-hids.tar.gz github.com/ossec/ossec-hids/
archive/2.9rc4.tar.gz
tar -xf ossec-hids.tar.gz
```

While downloading, verify that it is a signed commit in the OSSEC repo. You might have noticed that I downloaded the RC build for OSSEC, instead of the stable version. The reason for this is that the recent "stable" OSSEC version (v2.8.3) was released in Oct. 2015 and is considered a bit outdated now. Especially for today's agile environments, you want to start with the HIDS setup that includes all the latest features and fixes available.

Go to the OSSEC directory and copy the following variables into preloaded-vars.conf:

```
cd ossec-hids-2.9rc4/etc/
```

In `preloaded-vars.conf`:

```
USER_LANGUAGE="en" # For english
USER_NO_STOP="y"
USER_INSTALL_TYPE="local"
USER_DIR="/var/ossec"
USER_ENABLE_ACTIVE_RESPONSE="y"
USER_ENABLE_SYSCHECK="y"
USER_ENABLE_ROOTCHECK="y"
USER_ENABLE_EMAIL="n"
USER_ENABLE_FIREWALL_RESPONSE="y"
USER_WHITE_LIST="NS1 NS2"
```

For NS1 and NS2, specify the IP addresses for your name servers. These preloaded variables help in carrying out silent OSSEC installation.

Now, install the OSSEC:

```
cd ossec-hids-2.9rc4/
./install.sh
make && make install
```

After the OSSEC installation completes successfully, start it with the following command:

```
/var/ossec/bin/ossec-control start
```

SYSCHECK CONFIGURATION

The OSSEC main configuration file is `/var/ossec/etc/ossec.conf` where you can update the `<syscheck>` section to monitor various files for unauthorized edits and get notified about them in realtime.

Make a backup of the main configuration file:

```
cp /var/ossec/etc/ossec.conf /var/ossec/etc/ossec.conf-backup
```

Now you can go ahead with your configuration changes:

<DIRECTORIES>

This section is used to add or remove directories from OSSEC monitoring. It also has options to enable alerting for realtime changes.

```
<directories report_changes="yes" realtime="yes" check_
all="yes">/etc,/usr/bin,/usr/sbin</directories>
<directories check_all="yes">/bin,/sbin</directories>
```

The `report_changes` attribute reports changes (limited to text files for now) and real-time attribute generates notifications in real time.

<FREQUENCY>

The default frequency for full scans is 6 hours, i.e. 21,600 seconds, but you can change this to 1 hour or 24 hours—the scan itself is performed slowly to avoid any adverse impact on CPU/memory.

<ALERT_NEW_FILES>

By default, this option is set to "no." You can set this to "yes," and get notified when new files are created. However, this option only works during full scans that are performed at the frequency specified by you.

<AUTO_IGNORE>

Another attribute that is set to "no" by default. This option is kept disabled to avoid sending out large numbers of unnecessary notifications in case of misconfiguration. You can enable this option to receive a notification for all file changes as they happen. If disabled, then the OSSEC automatically ignores file changes after a 3rd change.

RULE CONFIGURATION

The OSSEC does not send notifications for new files by default. In order to enable this, you need to overwrite the corresponding rule in `local_rules.xml`.

Make a backup of `local_rules.xml` and add the following configuration block at the end (it should be present within the `<group>...</group>` section):

```
cp /var/ossec/rules/local_rules.xml /var/ossec/rules/local_rules.
xml-backup

<rule id="554" level="8" overwrite="yes">
  <category>ossec</category>
  <decoded_as>syscheck_new_entry</decoded_as>
  <description>File added to the system.</description>
  <group>syscheck,</group>
</rule>
```

JSON OUTPUT CONFIGURATION

The biggest advantage of selecting RC build over a stable version is that it supports the JSON output format for alerts generated. You can enable JSON output as follows:

```
<ossec_config>
  <global>
    <jsonout_output>yes</jsonout_output>
    ...
  </global>
  ...
</ossec_config>
```

The alert file will be stored at the following location:

```
/var/ossec/logs/alerts/alerts.json
```

This file can be sent either to a custom Elasticsearch server, or any managed centralized logging solution, such as Loggly. If you are using a managed centralized logging solution, you will also have the ability to generate a custom email notification on specific changes, or a specific set of servers. This approach significantly reduces non-essential email traffic.

SLACK NOTIFICATION CONFIGURATION

Add the following configuration in `ossec.conf` to enable Slack notifications:

```
<command>
  <name>ossec-slack</name>
  <executable>ossec-slack.sh</executable>
  <expect></expect> <!-- no expect args required -->
  <timeout_allowed>no</timeout_allowed>
</command>
<active-response>
  <command>ossec-slack</command>
  <location>local</location>
  <level>6</level>
</active-response>
```

By setting the `<level>` attribute to your desired value, you can filter out notifications and continue to receive only high-priority incidents on Slack. Also, you need to update `ossec-slack.sh` located at `/var/ossec/active-response/bin` with the following parameters:

```
SLACKUSER="ossec-alerts"
CHANNEL=""
SITE=""
```

The `SLACKUSER` is a custom name you would like to give to OSSEC alert notifier.

The `CHANNEL` specifies the name of a channel (public) or a group (private). If it is a group, then directly mention its name, e.g. ossec-alerts. If it is a channel, then you will need to include a hashtag (#) in the name, e.g. v#ossec-alerts.

The `SITE` will specify a URL to the incoming Slack webhook. You can configure an incoming Slack webhook here.

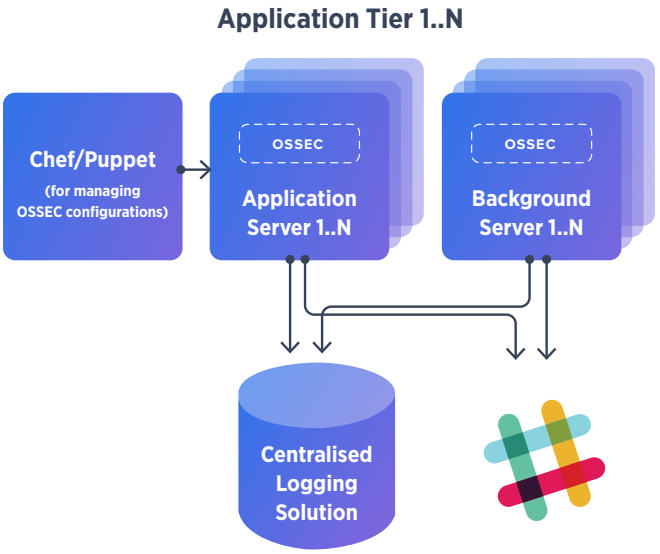
To have the configurations take effect, restart OSSEC with the following command:

```
/var/ossec/bin/ossec-control restart
```

PROPOSED ARCHITECTURE

The proposed architecture for OSSEC with a local model utilizes the tools that are generally available nowadays in agile environments. This approach also gives an ability for admins to receive selective notifications through Slack or email, and yet keep track of all events in a centralized place, which can be accessed with greater ease than perusing email. The configuration changes to OSSEC can be managed through deployment tools such as Chef or Puppet.

The architecture then looks like this:



This proposed architecture for OSSEC leverages the utilities that many enterprises already have or can easily deploy today, including a centralised logging solution and a company-wide communications platform such as Slack or Cisco Spark. The implementation described focuses on a distributed architecture, which is becoming the norm as applications grow. By deploying this solution to a large number of servers, OSSEC admins can receive important event notifications in an efficient, targeted manner without being inundated with irrelevant messages.

GAURAV PURANDARE is Principal Architect at Built.io, a technology provider with solutions that enable organizations to quickly create, integrate, and scale apps across web, mobile, and IoT. His passion for architecting cloud applications pairs with his years of experience managing and operating multi-tier, highly available applications. Gaurav is a certified AWS Solution Architect – Associate level and also holds certified MPN Competency for Microsoft Azure.

Adapting Serverless Architecture

QUICK VIEW

01 The evolution of Serverless Architecture

02 How Serverless Architecture works

03 Serverless Frameworks available today

04 Best practices for designing Serverless Functions

BY **IMESH GUNARATNE**

ASSOCIATE DIRECTOR/ARCHITECT AT **WSO2**

The serverless architectural style challenged the status quo of software design and deployment fundamentals for achieving the optimal development, operational, and management overhead. While it inherits elementary concepts from Microservices Architecture (MSA), it has been endowed with bleeding edge architectural patterns for attaining the minimum level of hardware idling possible. Despite the remarkable advancements it adds, adapting would need a thoughtful process for precisely mapping enterprise solution requirements to serverless computing.

The initial implementations of software systems, which were deployed on physical servers, could not optimally utilize the computation power of the underlying hardware since there could only be one instance of the operating system running at a given time. Later, after identifying the time-sharing capabilities in computing resources, multiple virtual computers were able to run on the same hardware concurrently by switching CPU and I/O operations among them. This technological evolution lead to many innovations in the industry and most importantly to the inception of the cloud. At this time, virtual machines were the most manageable, scalable, and programmable units of isolated computing environments for deploying software. Linux container technology emerged around the year 2006, when Google implemented control groups in line with the

FIGURE 1: EVOLUTION OF INFRASTRUCTURE

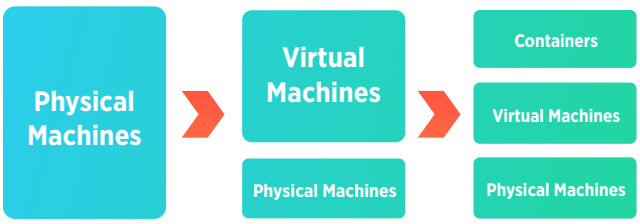
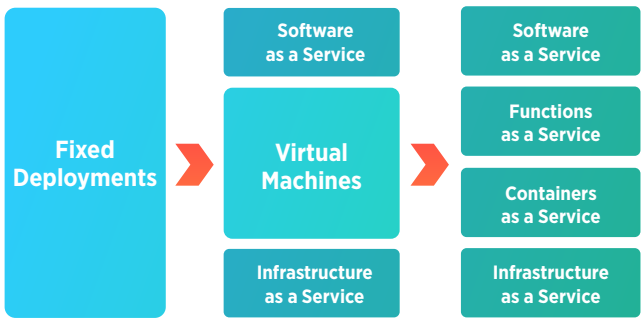


FIGURE 2: EVOLUTION OF SOFTWARE DEPLOYMENT METHODOLOGIES



Linux kernel features. Linux containers have been there ever since. However, only large scale, technically transcendent enterprises such as Google were able to use it at scale. Around the year 2012, the concept of the microservices architecture was introduced by a group of software architects in Europe. Later in the year 2013, Docker blazingly filled the gaps of accessibility, usability, and supporting services in the container ecosystem, and consequently containers started to become popular.

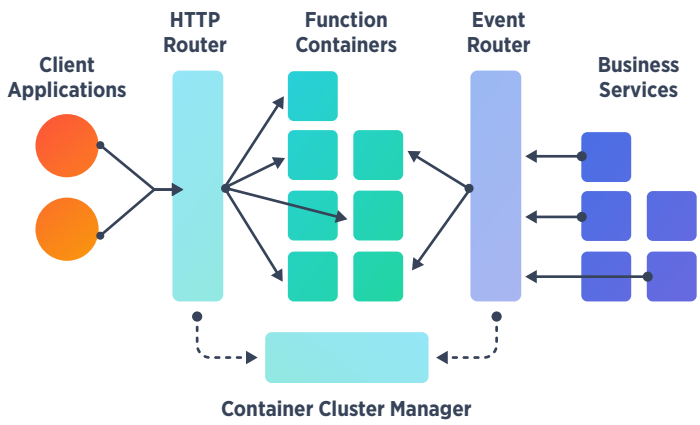
Linux containers opened up a new horizon for decomposing large monolithic systems into individual, self-contained services and executing them with fine-grained resource utilization. Expediting these advancements, container cluster management systems such as Kubernetes and Mesosphere started to rise in the same period for providing end-to-end Container as a Service (CaaS) capabilities. Later in 2015, AWS took another leap ahead by introducing AWS Lambda, which could further save software deployment costs by running microservices on demand and stopping them when there is no load. This concept is analogous to the stop-start feature in energy efficient vehicles which automatically shuts down the combustion engine for reducing fuel consumption.

HOW DOES IT WORK?

Despite the term "serverless" being nonsensical at first glance, its actual meaning is based on the ability to deploy software without having any involvement with infrastructure. Serverless platforms automate the entire process of building, deploying, and starting services on demand. Users would only need to register the required business functions and their resource requirements. As it is apparent, such functions can be classified into two main types: functions that get triggered by client requests and functions that need to be executed in the background for time triggers or events. Generally such serverless system can be implemented using a container cluster manager (CCM) with a dynamic router which could spin containers on demand. Nevertheless, it would also need to consider the latency of the router, container creation time, language support, protocol support, function interfaces, function initialization time, configuration parameter passing, providing certificate files, etc.

Even though this deployment style requires containers to be stopped when there is no load, in reality terminating containers so soon after serving requests would be costly, as there could be more requests coming in within short time intervals. Therefore more often, in serverless computing containers will get preserved for a preconfigured period of time to be reused for serving more requests. This is analogous to autoscaling behavior in PaaS platforms. Once a service is scaled up, instances will get preserved for a certain time period for processing more requests without terminating them immediately.

FIGURE 3: SERVERLESS ARCHITECTURE OVERVIEW



SELECTING A SERVERLESS PLATFORM

Serverless platforms are available in three different flavors:

1. Functions as a Service (FaaS) solutions available on public clouds:
   - A. AWS Lambda
   - B. Microsoft Azure Functions
   - C. Google Cloud Functions
   - D. Webtask
   - E. Syncano
2. Serverless frameworks that run on both public and private data centers:
   - A. Fission - Runs on Kubernetes
   - B. Funktion - Runs on Kubernetes
   - C. Kubeless - Runs on Kubernetes
   - D. Galactic Fog Gestalt - Runs on DC/OS
   - E. IBM OpenWhisk - Runs on Docker
   - F. IronFunctions - Runs on Docker, Docker Swarm, Kubernetes

3. Wrapper frameworks which provide platform agnostic APIs or/and value additions to existing serverless frameworks:
    A. Serverless.com - Supports AWS Lambda, IBM OpenWhisk
    B. Apex - Supports AWS Lambda

The decision of choosing a serverless platform would first depend on the data center that the solution is going to be run on. Next its features, maturity, and most importantly vendor lock-in possibilities would need to be evaluated. Out of the public FaaS offerings, AWS Lambda and Azure provide fully fledged serverless offerings. Google Cloud Functions is still in the alpha stage and only accessible via invitation. Fission and Funktion are two popular serverless frameworks available for Kubernetes. IronFunctions is a relatively new platform independent serverless framework being developed by Iron.io. IBM OpenWhisk platform has been donated to the Apache Software Foundation and now in incubation. Moreover, the third flavor provides options for implementing multi-cloud serverless deployments and also value additions for existing public FaaS offerings.

BEST PRACTICES FOR DESIGNING SERVERLESS FUNCTIONS

Consider the following design principles when designing serverless functions:

- Follow single responsibility principle when defining the scope of the functions.
- Optimize function implementation to execute in milliseconds.
- Stick to stateless protocols to let functions scale seamlessly.
- Use external services for service discovery, state, and cache management.
- Use environment variables for reading configurations without relying on files.
- Avoid using the file system for data persistence since containers are ephemeral by design.

CONCLUSION

Technologies and design patterns around serverless architecture are at their infancy. Today, new applications and extensions of existing systems can be moved to this architecture effortlessly if there are no restrictions on implementing those features in supported programming models. At the same time, it might be vital to choose stateless protocols for message receiving channels since RPC protocols such as WebSocket, Protocol Buffers, AMQP, MQTT, and Apache Thrift might require the listeners to be active all the time. Moreover business requirements such as implementing coordination logic, function grouping for implementing business processes, managing distributed transactions, state, etc. might need third party services and tools. Serverless platforms which use container pools might introduce security vulnerabilities depending on how containers are reused. They may also not allow resources to be configured per function. It is quite important to understand all of these aspects when adapting serverless architecture for a production system. Since many frameworks have been already established, this might be the right time to do a POC and plan for an upcoming project to evaluate its advantages.

REFERENCES

[1] **Serverless Architectures,** martinfowler.com/articles/serverless.html
[2] **Serverless Computing,** en.wikipedia.org/wiki/Serverless_computing
[3] **AWS Lambda,** aws.amazon.com/lambda
[4] **Azure Functions,** azure.microsoft.com/en-us/services/functions
[5] **Google Cloud Functions,** cloud.google.com/functions
[6] **Webtask,** webtask.io
[7] **Syncano,** syncano.io
[8] **Fission,** fission.io
[9] **Funktion,** funktion.fabric8.io
[10] **Kubeless,** github.com/skippbox/kubeless
[11] **Galactic Fog Gestalt,** galacticfog.com/product.html
[12] **OpenWhisk,** openwhisk.org
[13] **IronFunctions,** github.com/iron-io/functions
[13] **Apex,** github.com/apex/apex
[14] **TNS Guide to Serverless Technologies,** thenewstack.io/tns-guide-serverless-technologies-best-frameworks-platforms-tools

IMESH GUNARATNE is an Associate Director/Architect at WSO2 and an Apache committer. WSO2 is an open source middleware company which delivers solutions for Integration, API Management, Identity Management, IoT and Analytics. Imesh consults WSO2 clients on implementing middleware solutions and currently specializes in container related technologies. He led WSO2 PaaS team on delivering WSO2 Private PaaS, Docker, Kubernetes and Mesos based solutions. Imesh currently researches on Serverless Architecture and contributes to WSO2 platform team.

Cloud-Native Middleware Microservices

with Netflix's Hystrix Circuit Breaker and Eureka / Consul for Service Discovery

BY **KAI WÄHNER**
TECHNOLOGY EVANGELIST AT **TIBCO**

QUICK VIEW

01 A serverless platform allows you to run your application including computing, storing, and networking without the need of spinning up and managing a single (virtual) machine.

02 Being able to focus on software development instead of operating a fleet of servers is the primary driver behind serverless.

03 Dividing your application into small functions is necessary when following a serverless approach.

Cloud-native microservices offer many benefits. You can develop, test, deploy, and maintain independent lightweight services. You can easily combine various technologies, including programming languages such as Java or Go, and tools like integration middleware.

However, as you do not build monoliths anymore, "that complexity has moved and [...] increased [to] the outer architecture" as Gartner states. For this reason, new design patterns (have to) emerge to solve the challenges of independent, distributed microservices.

Middleware is used to interconnect everything. This is even more important in microservice architectures, where you have even more independent, distributed, and scaled business logic. Thus, as I discussed already around two years ago: microservices might be the death of the Enterprise Service Bus, but integration and middleware (and other things like API Management or cloud native concepts) will be needed more than ever before for microservice architectures.

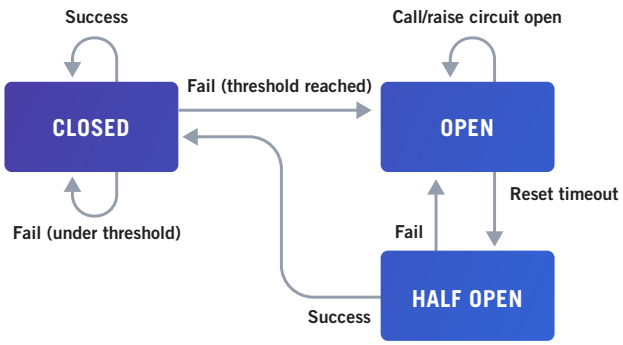
CIRCUIT BREAKER DESIGN PATTERN FOR RESILIENT MICROSERVICE ARCHITECTURES

The Circuit Breaker is one of these cloud-native design patterns. It allows for:

- Fast failing and rapid recovery
- Preventing cascading failures
- Latency tolerance logic
- Fault tolerance logic
- Fallback Options

This is realized by rejecting service requests if a service is not available for whatever reason; in a microservice architecture there can be many reasons or issues. The rejection is configured by various parameters such as request volume threshold or error threshold percentage.

Martin Fowler has a great explanation of the Circuit Breaker design pattern. Therefore, I will just explain it shortly using one of his graphics:



The circuit is closed in the beginning. All service request get a successful response from the service. If a threshold of 5 failures is reached, the circuit is opened. New service requests are rejected. After a timeout of 1 minute, a new service request tries if the service is available again; therefore the circuit is half open in this state. Depending on the success or failure, the circuit is opened or closed after this service request.

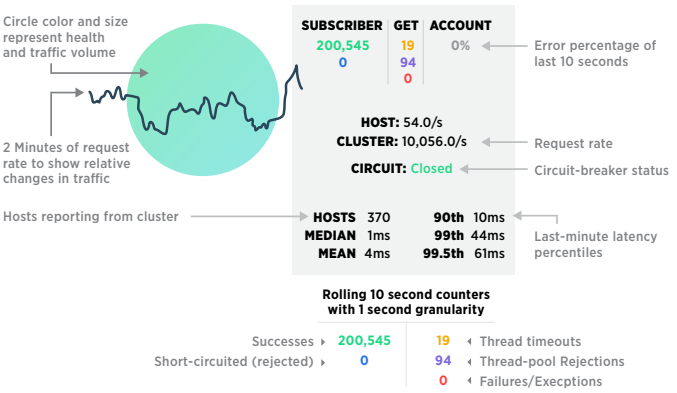
This relatively simple pattern can get very powerful (depending on the configuration options) and allows you to build resilient microservice architectures with reduced latency and lowered resource consumption.

NETFLIX'S OPEN-SOURCE IMPLEMENTATION 'HYSTRIX'

Hystrix was open sourced by Netflix a few years ago and is by far the most widely used framework for using the circuit breaker pattern in a microservices architecture.

Microservice architectures and cloud-native platforms such as CloudFoundry or Kubernetes can leverage Hystrix to build resilient

microservice deployments. In addition, you can use the Hystrix Dashboard to get some near real-time visualization. The video "Hystrix Dashboard - Tech Talk and Demo" shows a great introduction. Here is just a screenshot explaining the key aspects of the dashboard:



The dashboard does its job. But in a more sophisticated microservices architecture, you might leverage the benefits of a real-time streaming analytics visualization tool such as Zoomdata, Striim, or TIBCO Live Datamart. This allows not just monitoring live streaming data, but also applies continuous aggregations, rules, consolidation, and predictive analytics for automated or human-driven decision-making instead of just showing a monitoring dashboard for every microservice.

RESILIENT INTEGRATION MICROSERVICES WITH NETFLIX'S HYSTRIX AND MIDDLEWARE

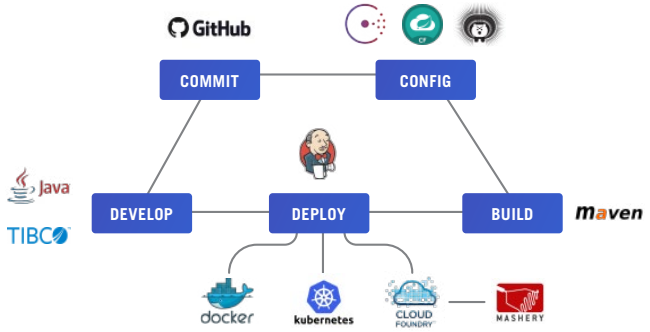
A resilient architecture is even more important for integration services because they interconnect everything. If the integration service is not resilient, fails all the time, or gets irresponsive, the complete enterprise gets into trouble. Therefore, circuit breakers can help a lot to make integration services more resilient.

A few middleware vendors already adopted cloud-native principles and offer native out-of-the-box support in their offerings. For example, Red Hat's Middleware Services run on the Kubernetes-based OpenShift PaaS. WSO2 is another open-source middleware vendor and already supports cloud-native platforms like Kubernetes. TIBCO offers BusinessWorks Container Edition (BWCE), which allows you to build, test, and package an integration microservice once, and deploy it everywhere, including Docker, Kubernetes, and CloudFoundry.

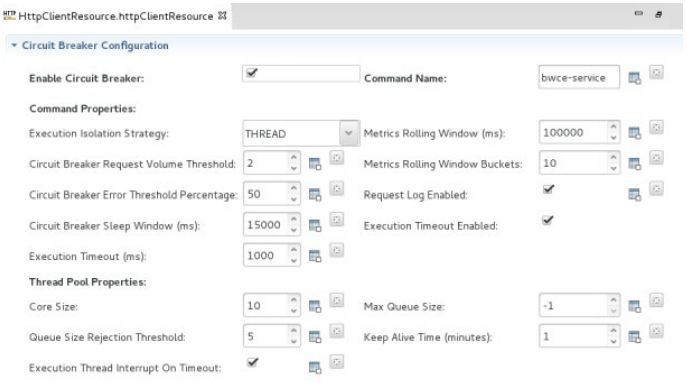
Note that cloud-native means much more than just putting your middleware into a Docker container! You need to support cloud-native concepts like circuit breaker or service discovery and integrate natively with these frameworks like Hystrix, Consul, or Eureka! If you just put your existing application or middleware software into a container, then you just "cloud wash" your solution and your architecture still looks like before, not leveraging cloud-native concepts.

The following shows the demo setup for the two videos recordings below. It includes several cloud native components (see top right).

With focus on Circuit Breakers, we use TIBCO BusinessWorks Container Edition (BWCE), Docker, and Netflix's Hystrix. The same could be achieved on other cloud-native platforms like Kubernetes or CloudFoundry in the same way.



BWCE offers out-of-the-box support for circuit breakers. You just enable it and configure the required parameters:



Details about the configuration and options can be found in the BWCE documentation. In the same way, you can also leverage service discovery frameworks like Consul or Eureka, and combine that with cloud-native platforms like Kubernetes or CloudFoundry.

DEVELOPMENT WITH TIBCO BWCE, HYSTRIX, CONSUL, KUBERNETES, AND CLOUD FOUNDRY

Watch a video recording to see how to use BWCE with Netflix's Hystrix Open Source Implementation of the Design Pattern 'Circuit Breaker' to develop, deploy and monitor cloud native middleware microservices. In another video you can see how to deploy an independent middleware microservice with Docker, Kubernetes, and Cloud Foundry without re-compiling, re-testing, or re-packaging.

In conclusion, we can say that cloud native design patterns are a very important piece of a resilient and scalable microservices architecture. This is even more true for middleware microservices, because these have to glue together all the other systems and business logic. Middleware is complementary to open-source implementations of these design patterns and should natively support them. Hystrix, Eureka, or Consul are just a few examples. Leverage whatever fits into your cloud platform and architecture.

KAI WÄHNER works as Technology Evangelist at TIBCO. Kai's main area of expertise lies within the fields of Big Data, Advanced Analytics, Machine Learning, Integration, Microservices, Internet of Things and Blockchain. He is regular speaker at international IT conferences such as JavaOne, O'Reilly Software Architecture or ApacheCon, writes articles for professional journals, and shares his experiences with new technologies on his blog (kai-waehner.de/blog).

Public vs. Private APIs

QUICK VIEW

01 Versioning a Web API is hard.

02 Private Web APIs are more commonplace than most developers realize.

03 We've been here before, with public and private APIs, and we can learn from the past.

BY **TODD FASULLO** DIRECTOR OF ENGINEERING AT **SMARTSHEET**

AND **TED NEWARD** DIRECTOR OF DEVELOPER RELATIONS AT **SMARTSHEET**

For those developers who are old enough to remember, it's with mixed parts nostalgia, disgust, and/or amusement that we recall the troubles Microsoft had with "undocumented APIs" back in the early 1990s. The subject of numerous conspiracy theories (and not a few books, including such icons as *Undocumented Windows* and *Windows Internals*, not to mention their nominal kin *Undocumented DOS* and others like them), the notion of an "undocumented" API call and its reasons for existence has seen much debate, discussion, and—so we thought—retirement. But if it's one thing age teaches us, it's that what's old is eventually new again.

Recently, debates began to rage around "APIs" and their legal status (looking at you, Google, and Oracle!). With the proliferation of Web APIs (looking at you, uh…. Internet!), the whole subject of "APIs" and their openness is starting to creep back upon us. More commonly, however, it seems that as more and more companies are moving to a REST-centric or -influenced style of architecture (particularly for mobile applications), companies are developing these Web APIs initially as something entirely for internal use, and only belatedly considering making them open to the rest of the world, usually after some kind of external or internal pressure to do so.

PRIVATE WEB APIS?

For many API developers, it's not clear what a "private Web API" would be or look like; to ensure that we're all on the same page, very simply, a private Web API would be an HTTP endpoint that isn't advertised to those who consume APIs (meaning developers, for the most part).

To be fair, for many developers, it doesn't start this way. The internal monologue usually begins with the realization that building a web application could/will end up having a mobile app (or two) as close kin, and that therefore some kind of centralized logic is necessary and desirable. "HTTP is ubiquitous," they nod sagely, "So let's use that." Before long, however, doubt creeps in: "Will this be something that others will want to consume? Will they consume it the same way as the web and mobile apps will? Perhaps the 'outside world' needs a simpler version of the API, but my internal needs are more complicated. Maybe the best path to take is to keep it obscured, at least to start."

Sometimes, the API implementors will take some step to make the API less discoverable, such as making it accessible only from machines inside a VPN, or restricting the IP range of acceptable incoming clients. In some cases, it's as simple as "If we don't tell them, they won't know." If the parameters to the API calls aren't described somewhere, nobody "unauthorized" to use them can, right?

Thus, on the surface, it would seem this discussion is entirely moot: so long as an API is accessible to callers, it would seem to be, by definition, accessible and therefore public. However, in an interesting twist, the same is true of the Windows environment thirty years ago: armed with only a few command-line tools (DUMPBIN.exe, shipping with every version of Visual C++/Visual Studio since 1992) and a basic knowledge of how the Portable Executable format is written (such as this article from 1994: (bit.ly/2l9tIme), any developer could discover method-entry points that weren't in the formal documentation set. Despite the ease with which an API could be discovered by anybody with basic knowledge of the system, Microsoft (and others) still left certain APIs to be undocumented and, presumably, untouched by their developer-consumers. In the 2017 era, those tools are

already bundled in every web browser and/or accessible via the command-line on most operating systems. HTTP, after all, is a far simpler standard than the Common Object File Format.

Why, then, might a company consider creating Web APIs, only to leave them unpublished?

SECURITY-THROUGH-OBSCURITY

Tackling the most common discussion first, although obscurity is never an appropriate replacement for security, simply not publishing the API will often keep 80% of the traffic away, since most developer-consumers simply don't want to depend on something that the company hasn't officially endorsed—and has obviously no plans to support.

And, to be fair, if the API is only exposed over a limited network space (such as a corporate intranet, hidden behind a firewall), then this approach seems completely adequate. However, practitioners of the "obscure API" should take note: only so long as that network remains inaccessible will the API remain obscure. As soon as partners are allowed into that network, or the company seeks to make APIs accessible over the public internet (for mobile clients, for example), the API is no longer obscure—a simple session with Wireshark (or any other TCP/IP-tracking network utility) will reveal its existence and open it up to use/attack.

COMPLEXITY

In some cases, the company seeks to protect the developer-consumer from having to know too much about the internal underpinnings behind the API. If the model, for example, is a complicated one that the API itself needs to honor, but is generally far more detailed than the user really needs in order to "get the work done," a company may segregate their APIs into two levels: a public API designed for the common case, and a less-public API designed to permit those who need that finer-grained level of detail.

Consider, for example, the Git source-control system. Part of the power of Git is its underlying data model, but for 99% of the users that use Git to keep versions of their source code up-to-date with one another, knowing that data model is incredibly unnecessary—so long as the tool abstracts things in terms of "files," most developers are perfectly comfortable with the higher-level abstraction, and never need to "drop down" to the lower-level APIs that work with that data model. (Git refers to this two-layer architecture as "the plumbing and the porcelain".)

Some might argue that this then calls for two separate APIs, one built in terms of the other. This would run afoul of the service-oriented nature of the Web API, though, which suggests that the service should never depend on anything other than itself. The reliability, for example, of any given service is only as great as the product of the reliability of all the services it depends on. (A service that depends on three other 99%-available services is thus 0.99 * 0.99 * 0.99, or 0.970299, which in turn doesn't take into account the reliability of the front-end service itself.)

VERSIONING

Probably one of the biggest reasons, then, to choose to leave an API "unpublished" is simply that versioning—or, to be more clear, versioning of the service over time—is a Really Hard Thing To Do(™). SemVer (semver.org), the semantic-versioning standard, offers up a standard way of stamping a service with version numbers—and more importantly, the discipline of managing the surface area of a published API. In theory, so the story goes, if developers follow this defined set of practices regarding the publication of version numbers, everything runs smoothly.

And yet, SemVer is difficult—it represents a discipline that many projects will not want to spend the energy taking into account, owing to the desire to drive to an MVP (Minimum Viable Product) to get it into the hands of developer-consumers, and iterate. In fact, an argument could be (and often has been) made that the SemVer rules shouldn't take effect until after the 1.0 release, when the company is ready to commit to the rules of a SemVer-governed project.

But in the meantime….

SUMMARY

As with most things software-related, the question of public-vs-private Web APIs defies easy binary solutions—there is no clear-cut demarcation. Faithful adherence to the governing REST-inspired ideal must be tempered with prudence and pragmatic concession to the reality of each situation.

We think it fair to assume that the default state for the Web API should be one that's publicized, documented, and supported. That said, it's becoming more and more clear that this default state is one that requires careful examination—for example, in a world of microservices, where each service is designed to do one-and-only-one thing, it becomes more likely that microservice-to-microservice dependencies are necessary, and that in turn will trigger questions around optimizing the communication between those services, followed by realizations of, "Do we really want any old client to be able to do some of these?"

Just bear in mind, whichever direction you go, you're trodding upon long-ago traversed ground. Take a moment to glance back Microsoft's way with a new-found appreciation for the decisions they made. And maybe, just maybe, find a good reason to pull out those old copies of *Undocumented Windows* that you just couldn't bring yourself to toss.

TODD FASULLO is a long time member of the Smartsheet Engineering team where he dedicates his time to designing and scaling Smartsheet's application for Smartsheet's millions of customers. When he's not improving the Smartsheet back-end infrastructure and development/build/testing infrastructure, Todd is often found tooling around the Pacific Northwest mountains with his family by bike, ski, or on foot.

TED NEWARD is the Director of Developer Relations at Smartsheet, and a long-time MVB at DZone; he currently resides in Redmond, WA, with his wife, two sons, cat, eight laptops, nine tablets, eleven mobile phones, and a rather large utility bill.

Executive Insights on Native Cloud Development

BY **TOM SMITH**
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01 Ultimately everything will reside in the cloud and integrate via APIs. The big question is when?

02 The most import elements of cloud development and deployment are scale, stability, and ease of use.

03 Ease of use, public cloud tools, and services are critical because we have a dearth of knowledgeable cloud developers, engineers, and DevOps professionals.

To gather insights on the state of cloud development and deployment today, we spoke with 15 executives from 13 companies that develop tools and services for companies to develop in, and deploy to, the cloud. Here's who we spoke to:

NISHANT PATEL, CTO, Built.io
GAURAV PURANDARE, Senior DevOps Engineer, Built.io
SACHA LABOUREY, CEO and Founder, CloudBees
JEFF WILLIAMS, Co-founder and CTO, Contrast Security
SAMER FALLOUH, V.P. Engineering, Dialexa
ANDREW TURNER, Senior Engineer, Dialexa
ANDERS WALLGREN, CTO, Electric Cloud
JACK NORRIS, S.V.P. Data and Applications, MapR
MICHAEL ELLIOT, Cloud Evangelist, NetApp
FAISAL MEMON, Technical Product Marketing, NGINX
SETH PROCTOR, CTO, NuoDB
PEDRO VERRUMA, CEO, rethumb
PETE CHADWICK, Director of Cloud Product Management, SUSE
NICK KEPHART, Senior Director Product Marketing, Thousand Eyes
DMITRY SOTNIKOV, V.P. of Cloud, WSO2

KEY FINDINGS

01 The solutions mentioned most often in developing in and deploying to the cloud is **Node.js and PHP,** followed by web services, AWS, and tools like Docker, Jenkins, Ansible, and Chef. A total of 32 different software and solutions were mentioned in addition to each of the interviewees producing software and solutions that their customers use to develop and deploy to the cloud.

02 The most important elements of cloud-based development and deployment are **speed, automation, integration, and scalability**. The big advantages to developing and deploying in the cloud are speed and agility, faster feedback to developers, and faster debugging. Defining the process and automating allows very fast development, deployment, and iteration, which enables developers to be more efficient and productive getting feedback in a timely fashion. The ability to deploy, back-up, and implement multi-cloud replication without sacrificing complexity or ease of management or increasing latency are all huge benefits. Finally, scalability and the ability to call for resources on-demand makes it easy to create and destroy resources on a consistent infrastructure while trusting your infrastructure can keep up with the workload generation and processing.

03 The problems being solved with cloud deployment and delivery are **scalability, stability, ease of use, and speed**. The key to faster, more reliable software development is a DevOps process in a cloud environment that ensures testing and security are continually monitored. Ultimately this enables developers and companies to make products and services that make the end user's life simpler and easier with a "fail fast, succeed fast" environment. The cloud has made developers more effective and efficient and forward-thinking companies realize it doesn't make sense to waste resources, managing hardware on something that's not their core competency.

04 Development and delivery in the cloud has evolved with the **tools and services that facilitate the automated and efficient delivery of quality code and apps**. Better

tools and more advanced services, including containers, enable developers to focus exclusively on their business logic. Cloud providers are doing a good job keeping up with the demand, making better and more flexible services available. The tools seen in existing enterprise frameworks can be mirrored to exist in the public cloud. This ensures that what is developed in one cloud is easily transferable to other environments. Elastic ephemeral computing is moving to data to ensure the reliability, scalability, and speed necessary to provide real-time operations and analysis to drive business value.

05 The primary obstacles to success to development and deployment in the cloud are**: 1) lack of experience and knowledge, 2) security, and 3) data.** There's a lack of knowledge and skills about the cloud and DevOps. The shortage of operational talent for development, deployment, and delivery make it difficult for companies and IT staffs to use all of the tools and services available because they're overloaded and have insufficient time to spend on projects and getting ahead of the ever-changing learning curve for the tools and services. Security continues to be an issue since the threat facing cloud-based development and deployment is significant. When an application moves to the cloud, the entire foundation and all of the security assumptions about the environment are different. This includes technology, people, processes, and the legal framework. The database can be an issue since the data must fulfill the SLA requirements around availability, security, disaster recovery, accessibility, and portability.

06 When asked if they had any concerns regarding development and deployment to the cloud, the most frequent response was "no," with virtually everyone seeing significant benefits to cloud development and deployment. Concerns that did exist revolved around **security and the lack of knowledgeable and trained cloud professionals**. Objections have been answered with regards to performance and security. Cloud vendors are providing DDoS mitigation services. There may be some concern with compliance with internal guidelines. The future of application security in the cloud is "self-protecting software" that can identify its own vulnerabilities and protect itself against attacks. If you don't have the expertise or proper training you are not as secure. You must manage the data effectively. There are a lot of tools and it's difficult for companies to keep up with all of the tools and to know which is best for their particular application.

07 The future of development and deployment in the cloud are around **integration, portability, speed, automation, and security**. We're in the midst of a digital transformation integrating platforms, customers, clients, and partners via APIs and a serverless architecture. More and more moves to the cloud until everything is living in the cloud. We're getting more agile delivering, and iterating, features more quickly to provide a superior user and customer experience (UX/CX). We'll use similar tools to span public and private clouds which will allow businesses to build workloads with independence—no vendor lock-in. Yet, we'll see an even greater concentration of power among the tier-one cloud platforms as they continue to build out tools and services that empower their customers with the ability to develop, deploy, and change platforms on demand.

08 Developers working on cloud-based projects needs to **keep the big picture in mind, learn the basics of the cloud infrastructure including the fundamentals of security in the cloud.** Spend time on different clouds to understand their nuances—they are not all the same. Think about which cloud platform is best to host your application, but use APIs to remain agnostic. Learn scripting and deployment environments like Chef, Puppet, and Ansible, and understand whether what you are building and running aligns with a container model. Only reinvent the wheel if you're going to make it better. 99 percent of what you're going to need is already out there. Think about the dependencies for your application to run, particularly with regards to an ever-expanding dataset. The first application you develop is not your last application. It will evolve over time and you need to be aware of how the changes affect latency and security. Security is a huge concern because the opportunity for devastating vulnerabilities increases exponentially in the cloud. Conduct security assessments continuously during the software development process. Use encryption keys to track who's issuing what and how often. Lastly, consider disaster scenarios, and how you will respond, once your application is in the cloud. Take holistic responsibility for your code.

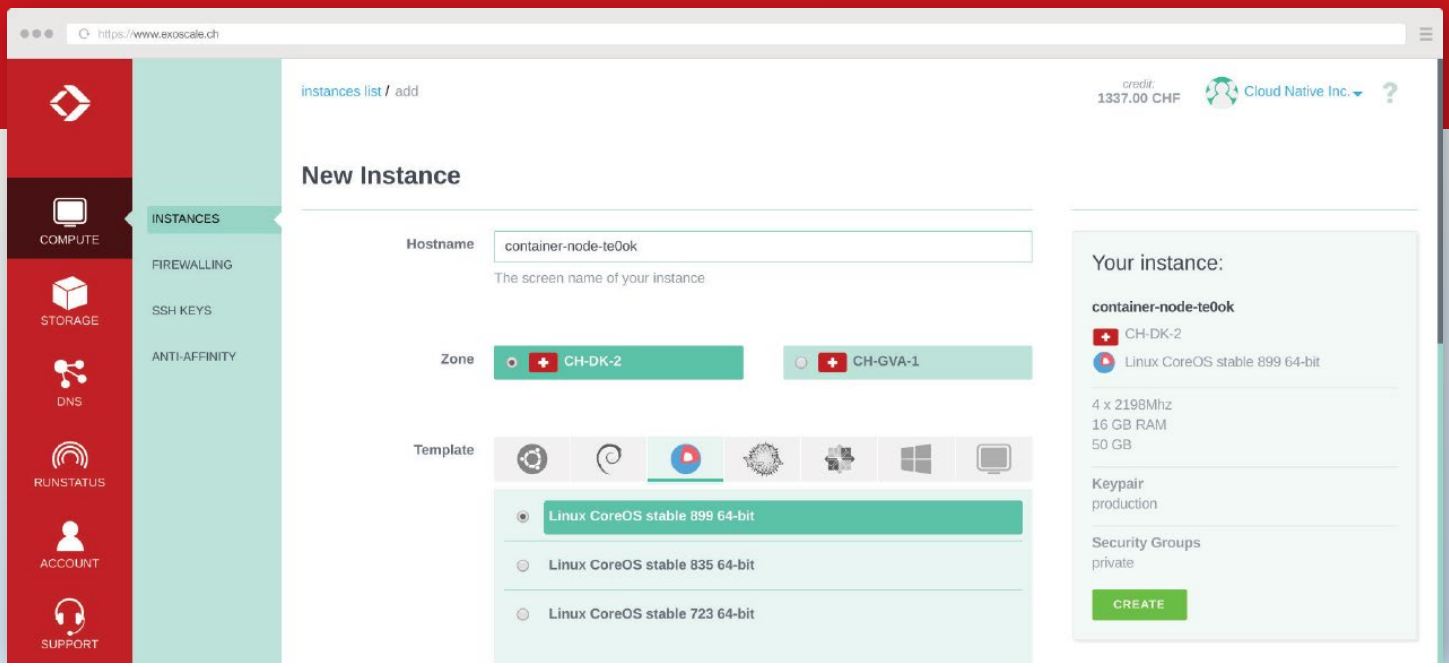
09 Additional considerations with regards to development and deployment in the cloud revolve around the **speed with which companies will move to the cloud, if they'll move everything or maintain a public and private cloud, and how they will continue to test and service their apps after deployment** to ensure they remain secure while delivering an excellent UX and CX.

- Organizations need to think about how they'll secure their code and other intellectual property.
- At what rate will we see legacy enterprise move to the cloud—including government and financial services?
- Will we see public clouds for development and private clouds for production?

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

exoscale

Simple cloud computing for cloud native teams.



Compute

- ✓ High performance SSD cloud servers
- ✓ Full 10 Gbit secure private networking
- ✓ Self-sustained zones for resilient deployments

Object Storage

- ✓ Highly available multi-redundancy object storage
- ✓ Low latency, high bandwidth public or private secure HTTP(s) access
- ✓ S3 compatible API for simple tooling integration

GPU Servers

- ✓ From 1 up to 4 Tesla P100 GPU cards
- ✓ Direct passthrough access for maximum performance
- ✓ All the advantages of a regular Compute server

DNS

- ✓ Anycast DNS network for low latency resolution
- ✓ Geo-replicated redundancy for optimal uptime

Runstatus

- ✓ Hosted status dashboards for devops teams
- ✓ Transparency for service incidents and scheduled maintenances

www.exoscale.ch

Solutions Directory

This directory of cloud platforms, tools, and services provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions are selected for inclusion in the directory based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

COMPANY NAME	PRODUCT	CLASSIFICATION	PUBLIC/PRIVATE	FREE TRIAL	WEBSITE
Adaptris	Interlok	iPaaS	Public	Available by request	adaptris.com/pages/products-and-services/interlok
Amazon Web Services	Amazon EC2	Virtual Machines	Public	750 hours, Limited by storage	aws.amazon.com/ec2
Amazon Web Services	AWS Elastic Beanstalk	Application Containers, aPaaS	Public	Limited by storage	aws.amazon.com/elasticbeanstalk
Anypresence	Anypresence	aPaaS, mBaaS	Public, Private by provider, Private on-premise	Pilot program available	anypresence.com
Apprenda	Apprenda	App Containers, aPaaS, DBaaS, MBaaS	Private on-premise, Hybrid	Limited by storage	apprenda.com
AppScale	Marketplace	Cloud Infrastructure Management	public, private, hybrid	Available by request	appscale.com
Armor Defense, Inc.	Armor Complete	Virtual Machines	Private	Available by request	armor.com/security-solutions/armor-complete
AT&T	AT&T Cloud Solutions	iPaaS, CDN	Public, On-premise	Available by request	synaptic.att.com
Baasbox	Baasbox	mBaaS	Public, Private on-premise	30 days	baasbox.com
Backendless	Backendless	mBaaS	on-premise, private	Available by request	backendless.com
CA Technologies	CA Cloud Solutions	Hybrid Cloud, SaaS, Cloud Security	Hybrid	Available by request	ca.com/us/why-ca/hybrid-cloud-solutions.html
Celigo	Celigo	iPaaS	Public	Available by request	celigo.com
CenturyLink	AppFog	App Containers, aPaaS, DBaaS	Public	7 days	ctl.io/appfog
CenturyLink	CenturyLink Cloud	Virtual Machines, App Containers	Public, Private by provider, Hybrid	30 days	ctl.io
ClearObject	ClearObject	Virtual Machines	Private	Available by request	clearobject.com
Clever Cloud	Clever Cloud Enterprise	App Containers, aPaaS, DBaaS	Any	No free trial	clever-cloud.com/enterprise

COMPANY NAME	PRODUCT	CLASSIFICATION	PUBLIC/PRIVATE	FREE TRIAL	WEBSITE
Cloud 66	Cloud 66	iPaaS, Container Management, CI	On-premise	Available by request	cloud66.com
Cloud Elements	Cloud Elements	iPaaS	Public, Private, Hybrid	Available by request	cloud-elements.com
CloudBees	Jenkins Platform	CI as a Service	Private, On-premise	14 days	cloudbees.com/products
Datapipe	Datapipe	Virtual Machines	Private by provider	None	datapipe.com/cloud
Dell	Boomi	iPaaS	Public	Available by request	boomi.com
Dell EMC	Dell EMC	Virtual Machines	Public, Private by provider, Hybrid	None	emc.com/en-us/cloud/hybrid-cloud-computing
DigitalOcean	DigitalOcean	Virtual Machines	Public	None	digitalocean.com/products/
Dimension Data	Dimension Data Cloud Solutions	Virtual Machines, App Containers	Public, Private	Available by request	dimensiondata.com/Global/Solutions/Cloud
DivvyCloud	DivvyCloud	Cloud Infrastructure Automation	Public, Private	Available by request	divvycloud.com
Docker	Docker Swarm	Container management	Any	Open Source	docker.com/products/docker-swarm
Engine Yard	Engine Yard	App Containers, aPaaS	Public	500 hours	engineyard.com
Exoscale	Exoscale	App Containers, aPaaS, DBaaS	Public, Private on-premise	14 days	exoscale.ch
FatFractal	FatFractal	Virtual Machines, App Containers, aPaaS, DBaaS, MBaaS	Public, Private on-premise, Hybrid	30 days	fatfractal.com
Flowgear	Flowgear	iPaaS	Public	14 days	flowgear.net
Fujitsu	Fujitsu Cloud Platform as a Service	iPaaS	Public, Private, On-premise	Available by request	fujitsu.com/global/services/application-services/paas
Fujitsu	Fujitsu Cloud Service K5	Virtual Machines	Public, Private	Available by request	fujitsu.com/global/solutions/cloud/k5/iaas
Gnubila	G Cloud Application Platform	aPaaS, DBaaS, iPaaS	Public	Available by request	gnubila.com/geas/g-cloud-application-platform
Google	Google App Engine	App Containers, aPaaS	Public	None	cloud.google.com/appengine
Google	Google Compute Engine	Virtual Machines	Public	None	cloud.google.com/compute
Heroku	Heroku	App Containers, aPaaS, DBaaS	Public	Limited by storage and 8 instances	heroku.com
HOSTING	Hosting Unified Cloud	Virtual Machines	Any	None	hostingunifiedcloud.com
HPE	HPE Helion Stackato	Virtual Machines, App Containers, aPaaS, DBaaS	Private by provider, Private on-premise, Hybrid	None	hpe.com/ca/en/software/multi-cloud-platform.html
HyperForm	HyperCloud	aPaaS, Container Management	Public, Private	Free tier available	hypergrid.com/technology

COMPANY NAME	PRODUCT	CLASSIFICATION	PUBLIC/PRIVATE	FREE TRIAL	WEBSITE
IBM	Bluemix	Virtual Machines	Private, Hybrid	Available by request	ibm.com/cloud-computing/bluemix
IBM	Bluemix	Virtual Machines, App Containers, aPaaS, iPaaS, DBaaS, MBaaS	Public, Hybrid	30 days	ibm.com/cloud-computing/bluemix
IBM	Softlayer	Virtual Machines	Any	30 days	softlayer.com
Informatica	Informatica	iPaaS	Public, Hybrid, On-premise	30 days	informatica.com/products/cloud-integration.html
Internap	Internap cloud services	Virtual Machines	Public, Private by provider, Hybrid	None	internap.com/cloud
Interoute	Interoute's Cloud & Hosting Services	Virtual Machines	Public	Available by request	interoute.com/products-and-services/cloud-and-hosting
Iron.io	IronWorker	aPaaS, Container Management, iPaaS	Private	Available by request	iron.io/platform/ironworker
Jelastic	Jelastic	Virtual Machines, App Containers, aPaaS, iPaaS	Any	Two weeks with hosting partner	jelastic.com
Jitterbit	Jitterbit	iPaaS, Cloud Migration	Public, Hybrid, On-premise	30 days	jitterbit.com
Joyent	Joyent	Virtual Machines	Public, Private on-premise, Hybrid	Free tier available	joyent.com
Kii	Kii	mBaaS	Any	Free tier available	kii.com
Kinvey	Kinvey	mBaaS	Private by-provider, Private on-premise	Limited by storage	kinvey.com
Kony	MobileFabric	aPaaS, iPaaS, mBaaS	Public, Private on-premise, Hybrid	90 days	kony.com/products/mobilefabric
Krystallize Technologies	Krystallize Technologies	Cloud Performance Management	Public	Available by request	krystallize.com
Kubernetes	Kubernetes	Container management	Any	Open Source	kubernetes.io
Kumulos	Kumulos	mBaaS	Private by-provider	Free until app is deployed	kumulos.com
Linode	Linode	Virtual Machines	Public, Private by provider	None	linode.com
Lunacloud	Lunacloud	Virtual Machines, App Containers, iPaaS, DBaaS, Model-Driven PaaS	Public	None	lunacloud.com
Mendix	Mendix App Platform	aPaaS, DBaaS, Model-Driven PaaS	Any	Free tier available	mendix.com/application-platform-as-a-service
Microsoft	Azure	Virtual Machines, App Containers, aPaaS	Public	30 days	azure.microsoft.com
MIOsoft	MIOedge	aPaaS, Analytics-as-a-Service, iPaaS	Public, Private	Available by request	miosoft.com/products/mioedge
Mirantis	Mirantis OpenStack	Cloud Infrastructure Management	Any	Open Source	mirantis.com/software/openstack
MongoDB	Atlas	Cloud-Native Database	Public	Available by request	mongodb.com/cloud/atlas

COMPANY NAME	PRODUCT	CLASSIFICATION	PUBLIC/PRIVATE	FREE TRIAL	WEBSITE
Morpheus Data	Morpheus Data	Cloud Infrastructure Management	public, private, hybrid	Available by request	morpheusdata.com
Mulesoft	Anypoint Platform	iPaaS	Public, Hybrid, On-premise	Available by request	mulesoft.com/platform/enterprise-integration
NaviSite	NaviSite	Virtual Machines	Public, Private on-premise	None	navisite.com
New Relic	New Relic APM	Application Performance Monitoring	Public	14 days	newrelic.com/application-monitoring
NGINX	NGINX Plus	Virtual Machines	Public, On-premise	30 days	nginx.com/products
NTT Communications	NTT Com Cloud	Virtual Machines	Private	Available by request	us.ntt.com/en/services/cloud.html
NuoDB	NuoDB	elastic SQL database	Hybrid	Available by request	nuodb.com
OpenStack	OpenStack	Virtual Machines	Public, Private on-premise	Open Source	openstack.com
Oracle	Oracle Cloud	aPaaS, DBaaS, iPaaS, App Containers, Virtual Machines	Any	Available by request	cloud.oracle.com
OrangeScape	KiSSFLOW	aPaaS	Public, Private, On-premise	Available by request	kissflow.com
Outsystems	Outsystems Platform	aPaaS, iPaaS, DBaaS, MBaaS, Model-Driven PaaS	Any	30 days	outsystems.com
Peak 10	Peak 10 Cloud Services	Virtual Machines	Public, Private	Available by request	peak10.com/products-services/cloud-services
Pivotal	Pivotal Cloud Foundry	App Containers, aPaaS, DBaaS	Public, Private on-premise	90 days	pivotal.io/platform
ProfitBricks	ProfitBricks	Virtual Machines	Public, Private by provider	14 days	profitbricks.com
Progress	Rollbase	aPaaS, Model-Driven PaaS	Any	30 days	progress.com/rollbase
Rackspace	Rackspace Cloud	Virtual Machines, DBaaS	Any	Free tier available	rackspace.com/cloud
Red Hat	Red Hat JBoss Enterprise Application Platform	App Containers, aPaaS	Any	Open Source	redhat.com/en/technologies/jboss-middleware/application-platform
Red Hat	Red Hat OpenStack Platform	Virtual Machines, App Containers, DBaaS	Public, Private on-premise	90 days	redhat.com/en/technologies/linux-platforms/openstack-platform
Redis Labs	Redis Cloud	Cloud Performance Management	public, private, on-premise, hybrid	Available by request	redislabs.com
RISC Networks	CloudScape	Cloud Migration Management	Public	Available by request	cloudscape.riscnetworks.com
Salesforce	Salesforce1	App Containers, aPaaS, iPaaS, DBaaS, Model-Driven PaaS	Public	Limited by storage	salesforce.com/solutions/mobile/overview
SAP	SAP HANA Cloud Platform	Virtual Machines, App Containers, aPaaS, iPaaS, DBaaS, MBaaS	Public	30 days	hcp.sap.com

COMPANY NAME	PRODUCT	CLASSIFICATION	PUBLIC/PRIVATE	FREE TRIAL	WEBSITE
Scality	Scality Ring	Cloud Storage	Public, Private	Available by request	scality.com/ring/object-storage-overview
ServiceNow	ServiceNow Cloud Management	Cloud Infrastructure Management	Public, Private	14 days	servicenow.com/products/orchestration/cloud-management.html
SingleHop	SingleHop	Virtual Machines	Public, Hybrid	Available by request	singlehop.com
SnapLogic	SnapLogic	iPaaS	Public, Hybrid, On-Premise	Available by request	snaplogic.com
Software AG	webMethods	aPaaS, iPaaS	Private, On-Premise	Available by request	softwareag.com/corporate/products/cloud/integration/default.asp
SUSe	SUSE	IaaS	Private	60 days	suse.com
Sysdig	Sysdig Cloud	Cloud Monitoring	On-premise	14 days	sysdig.com
Tangible	Xervo	aPaaS, Container Management, Cloud Infrastructure Management	Public	Available by request	xervo.io
TerraSky	SkyOnDemand	iPaaS	Public, Hybrid, On-Premise	Available by request	terrasky.com/products/skyondemand
The Apache Software Foundation	Apache CloudStack	Virtual Machines	Public, Private by provider	Open Source	cloudstack.apache.org
ThousandEyes	Cloud Agents	Cloud Monitoring	Any	Available by request	thousandeyes.com
Verizon	Verizon Cloud Solutions	Virtual Machines	Any	None	verizon.com
Virtusstream	Virtustream Cloud	Virtual Machines	Private by provider, Hybrid	None	virtustream.com/cloud
Vmware	vSphere	Virtual Machines, Cloud Infrastructure Management	Hybrid	Available by request	vmware.com/products/vsphere.html
Weaveworks	Weaveworks	Container management	Any	Available by request	weave.works
Windstream	Windstream	Virtual Machines	Public, Private on-premise, Hybrid	None	windstream.com
WorkXpress	WorkXpress	aPaaS, DBaaS, MBaaS	Public, Private on-premise, Private by-provider	30 days	workxpress.com
WSO2	App Cloud	Virtual Machines, App Containers, aPaaS, DBaaS, MBaaS	Public, Private on-premise, Hybrid	Free tier available	wso2.com/cloud/app-cloud
Youredi	Youredi	iPaaS	Public	Available by request	youredi.com
Zayo	Zayo	Virtual Machines	Public, Private, On-premise	30 days	zayo.com

GLOSSARY

APACHE THRIFT
An interface definition language and binary communication protocol.

API
Application Programming Interface, an endpoint exposed in a programming language that offers some useful feature or behavior.

CAP THEOREM
The idea that a distributed system can only provide two out of three benefits: consistency, availability, and partition tolerance.

CENTRALIZED LOGGING SOLUTION
Either a custom managed Elasticsearch-Logstash-Kibana (ELK) stack or a Software-as-a-Service (SaaS) solution. Having a centralized logging solution enables programmers or admins to easily view, compare, and correlate logs from different servers at the same place.

CIRCUIT BREAKER
A cloud-native design pattern to build and operate resilient, scalable microservices.

CLOUD-NATIVE APPLICATION
An application that can to take full advantage of a cloud environment (e.g. scalability, high availability).

CLOUD-NATIVE MIDDLEWARE
Middleware framework or product that natively leverages cloud-native concepts, design patterns, and cloud platforms.

CO-LOCATION
A data center that provides rental space, network connections, power, cooling, and security for servers that you manage and maintain.

DISTRIBUTED SYSTEM
Any number of computer systems linked by a network.

DYNAMIC OR AGILE ENVIRONMENT
An environment where servers are frequently scaled up or down.

EVENT-DRIVEN ARCHITECTURE
A pattern promoting the production and consumption of events used to integrate different parts of a system.

HOST-BASED INTRUSION DETECTION SYSTEM (HIDS)
A software application that monitors and analyzes a computer system for any unauthorized activity.

KUBERNETES
An open-source container cluster management platform maintained by Google.

MANAGED DNS
An external service provider that runs authoritative DNS servers on your behalf, answering queries about your domain names.

MESOSPHERE
A commercial container cluster management platform based on Apache Mesos.

MICROSERVICES
A pattern based on service-oriented architectures used to build cloud-native and independently deployable systems.

OPEN SOURCE HIDS SECURITY (OSSEC)
An Open Source Host-based Intrusion Detection System. It performs several functions, including log analysis, integrity checking, rootkit detection, real-time alerting, etc.

ORIGIN SERVER
Application servers that serve content to a CDN when an object is no longer cached or has expired.

PROTOCOL BUFFERS
Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data, similar to a smaller, faster, and simpler XML.

SEMANTIC VERSIONING
A governance scheme for how to structure a version number and when to adjust it.

SEMVER
A shorthand way to refer to "Semantic Versioning."

SERVERLESS
A platform providing computing, networking, and storage without the need of managing (virtual) machines.

SERVICE DISCOVERY
A cloud-native design pattern to discover distributed microservices in a flexible architecture.

WEB API
An HTTP endpoint designed to accept and return data, rather than HTML.

WEBSOCKET
A computer communications protocol, providing full-duplex communication channels over a single TCP connection.

INTRODUCING THE

Security Zone

Arm Yourselves From Hacks and Data Leaks!

Retroactively injecting security into applications is unnecessary and expensive. To avoid these costly patches, secure development has become essential for preventing hacks.

Our newest Zone addresses security issues head-on, offering tools, tutorials, and updates to secure your applications and the machines that run them!

NETWORK SECURITY

WEB APPLICATION SECURITY

DENIAL OF SERVICE ATTACKS

IoT SECURITY

dzone.com/security