CLOUD NATIVE JAVA JOSH LONG pdf

1: Cloud Native Java â€“ CoderProg

In this talk, we'll look at Spring www.enganchecubano.com to support modern microservices development, focusing on the things that really matter (or, at least, the things we've got cooking in Spring Boot

Sign up now and get the best price. The patterns for how we develop software, both in teams and as individuals, are always evolving. The open source software movement has provided the software industry with somewhat of a Cambrian explosion of tools, frameworks, platforms, and operating systemsâ€”all with an increasing focus on flexibility and automation. Known today simply as Amazon, the company now sells far more than just books. In , Amazon surpassed Walmart as the most valuable retailer in the United States. In the interview, Vogels talks about the core driver behind it. A large part of Amazon. He mentions that Amazon. Over time, as more and more teams operated on the same application, the boundaries of ownership of the codebase began to blur. Vogels pinpoints that shared resources such as databases made it difficult to scale out the overall business. The greater the number of shared resources, whether application servers or databases, the less control teams had when delivering features into production. You build it, you run it. He goes on to say that "the traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Many of the practices that Vogels spoke about in were seeds for popular software movements that are thriving today. Practices such as DevOps and microservices can be tied back to the ideas that Vogels introduced over a decade ago. While ideas like these were being developed at large internet companies similar to Amazon, the tooling around these ideas would take years to develop and mature into a service offering. The idea behind AWS was to provide a platform, the same platform Amazon used internally, and release it as a service to the public. Amazon was keen to see the opportunity to commoditize the ideas and tooling behind that platform. Many of the ideas that Vogels introduced were already built into the Amazon. The ideas behind the public cloud were sound. Virtual resources could be provisioned on-demand without needing to worry about the underlying infrastructure. Developers could simply rent a virtual machine to house their applications without needing to purchase or manage the infrastructure. This approach was a low-risk self-service option that would help to grow the appeal of the public cloud, with AWS leading the way in adoption. It would take years before AWS would mature into a set of services and patterns for building and running applications that are designed to be operated on a public cloud. While many developers flocked to these services for building new applications, many companies with existing applications still had concerns with migrations. Existing applications were not designed for portability. Also, many applications were still dependent on legacy workloads that were not compatible with the public cloud. In order for most large companies to take advantage of the public cloud, they would need to make changes in the way they developed their applications. The Promise of a Platform Platform is an overused word today. When we talk about platforms in computing, we are generally talking about a set of capabilities that help us to either build or run applications. Platforms are best summarized by the nature in which they impose constraints on how developers build applications. Platforms are able to automate tasks that are not essential to supporting the business requirements of an application. This makes development teams more agile in the way they are able to support only the features that help differentiate value for the business. Any team that has written shell scripts or stamped containers or virtual machines to automate deployment has built a platform, of sorts. How much work would it take to support the majority or even all of the requirements for continuously delivering new software? When we build platforms, we are creating a tool that automates a set of repeatable practices. Practices are formulated from a set of constraints that translate valuable ideas into a plan. What are our core ideas of the platform and why are they valuable? What are the constraints necessary to transform our ideas into practices? How do we automate constraints into a set of repeatable practices? At the core of every platform are simple ideas that, when realized, increase differentiated business value through the use of an automated tool. Werner Vogels stated that by increasing isolation between software components, teams would have more control over features they delivered into production. By increasing isolation between software components, we are able to deliver parts of the system both rapidly and independently. Constraints take the form of an opinion about how a core ideal will create value when automated into a practice. The following statements are opinionated constraints about how isolation of components can be increased. Software components are to be built as independently deployable services. All business logic in a service is encapsulated with the data it operates on. There is no direct access to a database from outside of a service. Services are to publish a web interface that allows access to its business logic from other services. With these constraints, we have taken an opinionated view on how isolation of software components will be increased in practice. The promises of these constraints, when automated into practices, will provide teams with more control over how features are delivered to production. The next step is to describe how these constraints can be captured into a set of repeatable practices. Practices that are derived from these constraints should be stated as a collection of promises. A self-service interface is provided to teams that allows for the provisioning of infrastructure required to operate applications. Applications are packaged as a bundled artifact and deployed to an environment using the self-service interface. Databases are provided to applications in the form of a service, and are to be provisioned using the self-service interface. An application is provided with credentials to a database as environment variables, but only after declaring an explicit relationship to the database as a service binding. Each application is provided with a service registry that is used as a manifest of where to locate external service dependencies. Each of the practices listed above takes on the form of a promise to the user. In this way, the intent of the ideas at the core of the platform are realized as constraints imposed on applications. Cloud native applications are built on a set of constraints that reduce the time spent doing undifferentiated heavy lifting. When AWS was first released to the public, Amazon did not force its users to adhere to the same constraints that they used internally for Amazon. Staying true to the name Amazon Web Services, AWS is not itself a cloud platform, but rather a collection of independent infrastructure services that can be composed into automated tooling resembling a platform of promises. Years after the first release of AWS, Amazon began offering a collection of managed platform services, with use cases ranging from IoT Internet of Things to machine learning. If every company needs to build its own platform from scratch, the amount of time delivering value in applications is delayed until the platform is fully assembled. Companies that were early adopters of AWS would have needed to assemble together some form of automation resembling a platform. Each company would have had to bake in a set of promises that captured the core ideas of how to develop and deliver software into production. More recently, the software industry has converged on the idea that there is a basic set of common promises that every cloud platform should make. The core motivation behind Cloud Foundry is to provide a platform that encapsulates a set of common promises for quickly building and operating. Cloud Foundry makes these promises while still providing portability between multiple different cloud infrastructure providers. The subject of much of this book is how to build cloud native Java applications. The Patterns New patterns for how we develop software enable us to think more about the behavior of our applications in production. Both developers and operators, together, are placing more emphasis on understanding how their applications will behave in production, with fewer assurances of how complexity will unravel in the event of a failure. As was the case with Amazon. Architectures are now focused on achieving ultra-scalability without sacrificing performance and availability. By breaking apart components of a monolith, engineering organizations are taking efforts to decentralize change management, providing teams with more control over how features make their way to production. By increasing isolation between components, software teams are starting to enter the world of distributed systems development, with a focus on building smaller, more singularly focused services with independent release cycles. Cloud native applications take advantage of a set of patterns that make teams more agile in the way they deliver features to production. As applications become more distributed a result of increasing isolation necessary to provide more control to the teams that own applications , the chance of failure in the way application components communicate becomes an important concern. As software applications turn into complex distributed systems, operational failures become an inevitable result. Cloud native application architectures provide the benefit of ultra-scalability while still maintaining guarantees about overall availability and performance of applications. While companies like Amazon reaped the benefits of ultra-scalability in the cloud, widely available tooling for building cloud-native applications had yet to surface. The tooling and platform would eventually surface as a collection of open source projects maintained by an early pioneer of the public cloud: Scalability To develop software faster, we are required to think about scale at all levels. Scale, in a most general sense, is a function of cost that produces value. The level of unpredictability that reduces that value is called risk. We are forced to frame scale in this context because building software is fraught with risks. The risks created by software developers are not always known to operators. By demanding that developers deliver features to production at a faster pace, we are adding to the risks of its operation without having a sense of empathy for its operators. The result of this is that operators grow distrustful of the software that developers produce. The lack of trust between developers and operators creates a blame culture: To alleviate the strain that is put on traditional structures of an IT organization, we need to rethink how software delivery and operations teams communicate. Communication between operations and developers can affect our ability to scale, as the goals of each party tend to become misaligned over time. To succeed at this requires a shift toward a more reliable kind of software developmentâ€”one that puts emphasis on the experience of an operations team inside the software development process; one that fosters shared learning and improvement. Reliability The expectations that are created between teams development, operations, user experience, etc. The contracts that are created between teams imply that some level of service is provided or consumed. By looking at how teams provide services to one another in the process of developing software, we can better understand how failures in communication can introduce risk that leads to failures down the road. Service agreements between teams are created in order to reduce the risk of unexpected behavior in the overall functions of scale that produce value for a business. A service agreement between teams is made explicit in order to guarantee that behaviors are consistent with the expected cost of operations.

2: Unamanic: An Unrepentant Geek

Authors Josh Long and Kenny Bastani fully immerse you in the tools and methodologies that will help you transform your legacy application into one that is genuinely cloud native. In four sections, this book takes you through.

3: Cloud Native Java

Josh is a Java Champion, author of 5 books (including O'Reilly's upcoming Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry) and 3 best-selling.

4: The cloud native application - O'Reilly Media

In this talk, we'll look at how high performance organizations like Ticketmaster, Alibaba, and Netflix make short work of that complexity with Spring Boot and Spring Cloud. Josh Long July 27,

5: Cloud Native Java [Book]

Glad to say that "Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry" bu Josh Long and Kenny Bastani returns to me to the comfort zone of O'Reilly quality.

6: Cloud Native Java : Books

Cloud Native Java. Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry Josh Long, Kenny Bastani. Learn what separates traditional enterprise software development from the likes of Amazon, Netflix, and Etsy.

7: Cloud Native Java - Voxxed

A collection of training courses and example code for Cloud Native Java Java 20 14 Apache Updated Jul 3, it-support.

CLOUD NATIVE JAVA JOSH LONG pdf

8: Cloud-native Java: Software Architecture Conference | Microservices training | O'Reilly

Josh Long is the Spring developer advocate, an editor on the Java queue for www.enganchecubano.com, and the lead author on several books, including Apress' Spring Recipes, 2nd Edition. Josh has spoken at many different industry conferences internationally incl.

9: The Cloud Native Java Book â€“ Home

Read "Cloud Native Java Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry" by Josh Long with Rakuten Kobo. Learn the essentials of the Spring Boot microframework for developing modern, cloud-ready JVM applications and microserv.

CLOUD NATIVE JAVA JOSH LONG pdf

Blessed be your glorious name The avitaminoses; the chemical, clinical and pathological aspects of the vitamin deficiency diseases Basic cosmetic chemistry k2s A dependent court A Biographical History of Greene County, Pennsylvania Aesthetics and nature The Seafarers of Catan (The Settlers of Catan) Maniacs Alice Elliiott Dark Sir Thomas Wyatt, a literary portrait Community arts centers Damage Assessment of Structures VII Android studio quiz app tutorial The 00 Lunar Calendar The Kind of Girl I Am Chronicles of the great rebellion. Emperor Penguins (Pull Ahead Books) Refugee Community Organisations And Dispersal Overheard while shopping Slimming world food diary Pharmacotheon entheogenic drugs their plant sources and history Pharmacology kd tripathi 7th edition Contracts in writing and third party Utah co-starring Tex Ritter, Horace Murphy, Charles King, Adele Pearce and Karl Hackett ; directed by Alb Stress and deformation Pahl and beitz engineering design a systematic approach Motor racing the Grand Prix greats The New American Standard Bible Writers map of Toronto The Evolution of Key Mass Communication Concepts If I Can, You Can State of illinois child support worksheet 2017 Project change request form Ron Ndabezitha Everett-Karenga. A. The Enlightenment The Athletics win another pennant Belshazzars Feast and the Fall of Babylon The Wars of Alexander the Great, 336-323 BC The Robin Hood Companion SECOND COUNCIL OF CARTHAGE UNDER CYPRIAN (A.D. 252 206 Manual ellipsis 10 tablet