GLOSSARY OF REACTIVE SOFTWARE SYSTEMS

At Xtiva, we are strong believers in the power of the cloud-based computing to dramatically improve complex technology and system architectures in the financial services industry.

Find out more about XtivaCloud, Xtiva's next generation software platform.

Reactive System Development is an exciting approach to building cloud-based systems that are flexible, loosely-coupled, secure and scalable, making them easier to develop and adapt to change. But if you don't spend all day looking at technology like we do, some of these terms might be a bit unfamiliar. That's why we've compiled some key terms to help you better understand Reactive systems at a glance.

API – SHORT FOR 'APPLICATION PROGRAM INTERFACE'

Basically, this is how different systems and apps can intelligently 'talk' to one another. Because the many different components in a Reactive architecture are self-contained, they rely on their ability to communicate with one another using a standardized protocol, such as APIs. This structure also facilitates openness and extensibility of a system into inter-operations with other disparate systems in the application space, or data eco-sphere of interest.

CONTAINERIZATION

An approach to virtualization that is seen by many as a more efficient method than strictly using virtual machines (VM). Each virtual machine includes its own embedded operating system (OS), whereas multiple containers can use a single OS install. Multiple containers can exist within a single VM. Also, it describes the way microservices are isolated from one another (i.e. in their own containers) that interact with one another, but operate independently.

CAPEX (VS OPEX)

Short for 'capital expenditure' which represents investments in assets that depreciate over time. Capital investments are usually separate from a company's operating budget, and this shift in budget allocation can create challenges when finance and operations need to reconcile their activities. The conversion of physical data centers to cloud-based operations has a potentially profound effect on the balance between Capex vs. Opex, since the former relies on very heavy hardware and related services budgeting with multi-year deprecation considerations (long Capex), while the latter virtually eliminates the need for capital expenditures, in favor of a much more Opex-centric spend (minimal Capex). The financial ramifications of this technology shift (from rigid physical hosting to reactive cloud hosting) is therefore an important business consideration.

CQRS (COMMAND QUERY RESPONSIBILITY SEGREGATION)

A way of implementing queries in a microservice architecture, which is asynchronous. Splitting up the responsibility between commands and queries, eg., "writing" (command) performs an action and "reading" (query) receives information without changing anything. Basically, it's the idea that you can use a different model to *update* information than the model you use to *read* information. This presents fundamental advantages in differential scaling, security and audit trail preservation, and user responsiveness.

DECOUPLING OR “LOOSE COUPLING”

Describes the modular nature of individual microservices. Isolation and containment allow each element to operate independently. This makes it easy to build and test new ideas quickly, and keeps failures from spreading. It also eliminates the age-old “Coral Reef Effect” of most legacy application architectures, which become very large and increasingly fragile to change (i.e. touch the coral and it dies, while you receive a painful cut – that’s legacy apps for you). In contrast, loose coupling means that new capabilities may be easily and frequently added without destabilizing or damaging any pre-existing functionality.

DEVOPS

This is one of the fastest-growing domains in software because it combines what once were two nearly opposing forces into a single, powerful, skill set: namely, Development vs. Operations. DevOps entails the simultaneous development of the IT Operating Environment itself, in perfect harmony with the development of the solution software. In many organizations, software development is evolving into this new form of traditional IT/Hosting Operations because the line between “Running the Environment” and “Building the Application” has become blurred, nearly meaningless, in the world of Reactive Systems. Now the entire environment is dynamic, not just the logical flows of data through the application. This new DevOps arena also demands even more effective collaboration and communications between Developers and Production Operations (specifically, IT Operations). This requires a fundamental mindset shift, since the production operations team can collaborate with the DevOps team and benefit from the Agile mindset of software developers. For example, embracing iterative versus ‘big bang’ releases, leveraging reactive recovery as opposed to traditional back-up-based DR, etc.

DOMAIN-DRIVEN DESIGN

In this context, a *domain* is ‘a sphere of knowledge, influence, or activity. The subject area to which the user applies a program is the domain of the software.’ Organizations using domain-driven design build teams, processes, and systems around intuitive domains like customer service, rather than its internal departments like IT and sales. So a customer service team might include a salesperson, a technology specialist and a customer service rep, and tools are naturally built around that customer domain. See Conway’s Law.

ELASTICITY (VS. SCALABILITY)

The ability to automatically scale down the resources used by the software system when the workload *decreases*, improving efficiency and decreasing cost, in addition to scaling up. There’s no elasticity without scalability.

ELASTICSEARCH

Highly scalable search engine optimized for storing, indexing and analyzing broad arrays of data without relying on a traditionally pre-defined Data Model (data tables, joins, etc.) as the starting point. Elasticsearch enables powerful data discovery and is well suits to the CQRS architecture because it acts as a search engine rather than a static repository, so differentiating between Reads & Writes comes naturally.

ERROR (VS. FAILURE)

A potential outcome that is fully expected and has been anticipated by the system’s architects, and which does not impair normal system operations, performance or stability. For example, a user may enter data in an incorrect format and receive an error message, with instructions to re-enter the data correctly. A properly designed system

will communicate the nature of the error to the user, making it straightforward to correct, while also defending the system from such errors for security and stability reasons.

FAILURE (VS. ERROR)

When a system ceases to work normally, often because of a hardware malfunction, resource exhaustion, or data corruption. Not all failures are fatal, but they usually result in decreased capacity at least temporarily, until they are repaired with human intervention. This is a topic for increasing application of machine learning techniques as well. The concept being that a separate intelligent agent can surveille a system under 'normal' operation and detect un-anticipatable failures (hardware or connectivity service issues, attacks, etc.) to gather information either to summon/inform support staff, or ultimately, to resolve/thwart the failure mode automatically.

IAAS (INFRASTRUCTURE-AS-A-SERVICE)

Provides firms with resources and capacity well beyond what would be cost-effective to build and maintain internally. Think about it; is it more cost effective to maintain +500% of the typical day's operating capacity at all times?, or to pay only for surge-capacity for the few hours per day (or month, week, year) when it's actually needed?

IAC (INFRASTRUCTURE AS CODE)

The automated process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. The inference here is that by transforming the infrastructure into a codable format, automated means may be employed to respond to dynamic scaling and other needs without requiring the extensive human intervention previously needed.

MESSAGE-DRIVEN (VS. EVENT-DRIVEN)

Reactive systems are message-driven (vs. event-driven), meaning that the sending and receiving of messages can happen independently in terms of time and space (asynchronous and location-independent). Here's a great explanation from Kevin Webber at Red Elastic:

Why is a message-driven architecture so important for responsiveness? The world is asynchronous. Here's an example: you're about to brew a pot of coffee, but you realize you're out of cream and sugar.

One possible approach:

- *Begin to brew a pot of coffee.*
- *Go to the store while the coffee is brewing.*
- *Buy cream and sugar.*
- *Return home.*
- *Drink coffee immediately.*
- *Enjoy life.*

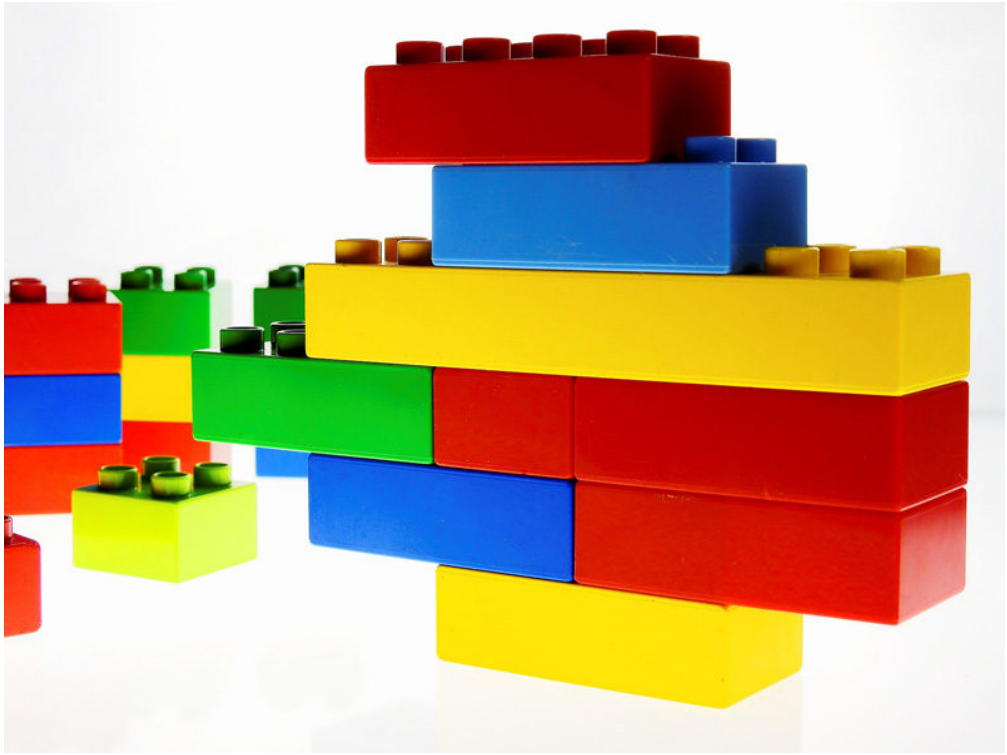
Another possible approach:

- *Go to the store.*
- *Buy cream and sugar.*
- *Return home.*
- *Start brewing a pot of coffee.*
- *Impatiently watch the pot of coffee as it brews.*
- *Experience caffeine withdrawal.*
- *Crash.*

As you can clearly see, a message-driven architecture provides you with an asynchronous boundary that decouples you from time and space.

MICROSERVICES

Generally, a specialized way of deploying software systems that is based on a network of processes that are flexible and modular. An individual microservice is: "an application with a single function, such as routing network traffic, making an online payment or analyzing a medical result." Often represented as a honeycomb or Lego blocks, microservices can be strung together into functional applications.



OPEX (VS. CAPEX)

Short for ‘operating expenditure.’ As firms move to cloud-based systems, their outlay shifts from capex (investing in physical assets like servers) to opex (ongoing costs to use infrastructure and software owned and maintained by 3rd parties).

RESILIENCE

One of the key tenets of the Reactive programming ethos, resilience means that the system reacts to failure in a productive way (instead of just failing altogether and ceasing to function). It is designed so that components are isolated and failures can be contained and overcome without affecting the overall responsiveness of the system.

RESPONSIVE

A responsive system responds to users in a timely manner and performs consistently. Responsiveness is the foundation of usability. It also means that problems are detected and dealt with quickly by the system in a way that is transparent to performance, while minimizing the need for human intervention.

SAAS (SOFTWARE AS A SERVICE)

A software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted.

SCALABILITY (VS. ELASTICITY)

The ability to adapt to a workload increase by provisioning the necessary resources. Scalability is a reaction to increased demands on the system, while elasticity is the ability to both increase and decrease the use of resources (‘provisioning’ and ‘deprovisioning’) automatically, depending on the workload.

USER

In Reactive systems, this term has multiple meanings. It could mean the human agent issuing commands to the software or the human agent accessing or 'reading' data from a system. It could also mean a non-human component of a system issuing commands or receiving information to perform its job.

VIRTUALIZATION

Software that allows hardware to behave like multiple distinct pieces of hardware. A single server can run multiple independent operating environments, each appearing to the user like its own dedicated server.

VIRTUAL MACHINE (VM)

Brian Kirsch explains succinctly: *"An operating system (OS) or application environment that is installed on software, which imitates dedicated hardware. The end user has the same experience on a virtual machine as they would have on dedicated hardware."*

Find out more about XtivaCloud, Xtiva's next generation software platform.

FOR MORE INFORMATION ON REACTIVE SOFTWARE SYSTEMS AND RELATED CONCEPTS:

- The Reactive Manifesto
- The Move from CapEx to OpEx in the Cloud (Cloud Cruiser)
- The CxO's Guide to Microservices (ThoughtWorks)
- Microservices 101 (Accenture)
- What is Reactive Programming? (Red Elastic)
- The introduction to Reactive programming you've been missing (Github)
- Developing Reactive Microservices (Markus Eisele, Safari Books)
- A Brief Overview of Reactive Systems (IDG Infoworld)
- What are Containers and Microservices? ComputerWeekly.com
- Virtualization is dead, long live containerization (Diginomica, 2014)
- Virtualization (Wikipedia)
- Path to microservices: Moving away from monolithic architecture in financial services (Devbridge)
- Finance IT: The future of microservices, DevOps and the cloud in banking systems (JAXenter)
- Microservices – The New Building Blocks of Financial Technology (Quantifi, 2016)
- [Video] Cloud and Microservices in Banking and Finance (The Realization Group)
- Why Microservices Architecture is so BIG for Today's Financial Services Industry (HP Enterprise)
- A Brave New World of Microservices (Fintech Weekly)
- What are Microservices? (Pivotal)