Microservice Maturity Model Proposal – Daniel Bryant (@danielbryantuk)

Name	Megalith Platform	Monolith Platform	Macro SOA Platform	Meso Application Platform	Microservice Platform	Nanoservice Platform
Description	Humongous single codebase resulting in a single application	Large single codebase resulting in a single application	Classical SOA applications, and platforms consisting of loosely-coupled large services (potentially a series of interconnected monoliths)	Meso' or middle-sized services interconnected to form a single application or platform. Essentially a monolith and microservice hybrid	Cloud native' loosely-coupled small services focused around DDD-inspired 'bounded contexts'	Extremely small single-purpose (primarily reactive) services
Motivations	These systems typically result from the implementation of an initial intensively planned complex application, which has evolved in a haphazard fashion over many years of success in the market	Monolithic applications initially allow rapid application development, and early features are added easily and quickly. When the application codebase is small the system is easy to understand, change and deploy	Macrolithic systems generally result from organisations with clearly defined logical groupings of business activity (i.e. risk analysis, billing, account management). There is often integration within the platform of multiple disparate systems, potentially acquired via merger and acquisition activity	Meso systems emerge from organisations migrating away from monolith, or adding new functionality to existing applications via externally hosted smaller applications	Microservice systems promote the single responsibility principle, and are potentially easier to understand and maintain even as the application grows in size and complexity. They can also enable decreased time to market (changes are isolated), and enable flexible scalability	Platforms such as AWS Lambda are allowing applications to be developed that are ideal for responding to simple event-driven systems with dynamic workloads. This can be thought of as an extension to the 'blackboard' architectural model, although state is stored externally
Challenges	Near impossible to understand, maintain and modify.	Difficult to understand, maintain and evolve	Difficult to reason about system-level behaviour. The platforms onto which these systems are deployed are typically controlled by a vendor, and there may be large licensing costs. Change typically involves a lot of organisation-level coordination.	These systems often contain the challenges of both the monolithic and microservice architectures. It can also be difficult to extract functionality from the monolith, and there will be changes required in development and deployment practices and tooling. Potentially two separate skillsets required.	Complexity pushed from applications to deployment and runtime orchestration. Multiple services also require resilient discovery and communication mechanisms. Services must be explicitly designed to allow flexible scalability and fault-tolerance	Due to the asynchronous and reactive nature of these platforms, they often display emergent behaviour that can be difficult to debug at the system level
Typical Codebase Age	10 years +	5 years +	5+ years	2 - 5 years +	New	New
Languages	Single	Single	Multiple	Multiple	Polyglot	Polyglot (currently limited)
KLOC Per Artifact	1000's	100's	10's	10 - 1's	1's	0.1's
Code-level Modularisation	Ball of mud	Ball of mud, potentially with some componentization (packages, namespaces, JARs, DLLs etc)	Typically componentized at the platform level, but individual service code may be less well organized and implemented according to, or coupled with, vendor APIs	Hybrid of monolith and microservice platforms	Highly-componentised	Extreme componentization
Typical Code Coupling	High	High	High - Medium	High - Low	Low	Extremely Low
Typical Code Cohesion	Low	Low - Medium	Medium - High	Medium - High	High	Extremely High
Typical Inter-application Communication	In-process	In-process	Out-of-process via heavyweight protocols and middleware, e.g. WS-*, SOAP, XML, ESB (TIBCO, Oracle Service Bus)	Out-of-process with a combination of lightweight middleware and protocols, e.g. ESB (Mule ESB, Fuse, WS02), HTTP, AMQP	Out-of-process, typically using lightweight protocols e.g. HTTP, Protocol Buffers, JSON, AMQP	Out-of-process, typically using lightweight protocols e.g. HTTP, Protocol Buffers, JSON, AMQP
State	Application typically stateful. Long-lived state typically persisted externally typically following ACID principles	Application typically stateful, with sticky sessions. Long-lived state persisted externally using ACID principles. Caching used extensively at the edge, and throughout stack	Combination of (sticky) stateful and stateless services. Long-lived state typically persisted externally	Hybrid of monolithic and microservice platform	Services typically stateless, with data persisted externally, often eventually consistent. Extensive use of caching	None
Data Stores	Single, external	Multiple, external	Multiple 'enterprise' data stores and middleware	Combination of monolith and microservice	Multiple, integrated with individual services	Single, external
Data Store Type	RDBMS, flat file	RDBMS, search indexes (e.g. Solr, ElasticSearch)	RDBMS, search indexes, data grids (e.g Coherence, Infinispan)	Combination of monolith and microservice	RDBMS, search indexes, NoSQL, lightweight data grids (e.g. Hazelcast)	NoSQL

Microservice Maturity Model Proposal - Daniel Bryant (@danielbryantuk)

Name	Megalith Platform	Monolith Platform	Macro SOA Platform	Meso Application Platform	Microservice Platform	Nanoservice Platform
Application Runtime Longevity	Eternal	Long-lived	Long-lived	Combination of long and short lived (depending on application)	Transient	Ephemeral
Scalability	Vertical	Initially vertical, and then horizontal via cloning / clustering and load balancing. Vertical scaling is typically not on-demand	Vertical and horizontal, depending on service and vendor offerings	Combination of monolith and microservice	Horizontal service-specific cloning / clustering (typically on-demand)	Horizontal (on-demand)
Deployment Platforms	Bespoke	Bespoke	Typically vendor-specific 'Enterprise' platforms	Combination of monolith and microservice platforms	IaaS, PaaS (private or public), or container cluster manager (e.g. Mesos, Kubernetes)	PaaS e.g. AWS Lambda
Deployment Fabric	Bare metal in a private data center	Bare metal or virtualized data center, or public / private cloud	Bare metal in a vendor managed data center or private/public cloud	Bare metal or virtualized data center, or public / private cloud	Public / private cloud with VMs or container-ready OS	Bespoke, typically supporting LXC / containers 'under the hood'
Deployment Artifacts	Single large artifact	Single large artifact	Multiple medium-large artifacts	Typically one / several large artifacts, and multiple smaller artifacts	Large number of small artifacts	Large number of tiny artifacts
Deployment Orchestration	Manual	Manual with potentially some scripting, or automated build pipeline implemented via CI tooling (Jenkins, MS TFS) in combination with release trains code deployments	Tooling typically provided by vendor platform	Semi-automated orchestration with multiple build pipelines (typically a hybrid of monolith and microservice platform approaches)	Automated build pipelines implemented via CI tooling (Jenkins, MS TFS). Additional tooling may be required to automate deployments (e.g. Netflix's Asgard)	Complexity varies depending on number of services and vendor platform
Infrastructure Mutability	Snowflakes, with servers treated as sacred pets	Servers treated as pets	Bespoke hardware configuration	Move towards immutable infrastructure	Immutable 'Phoenix' servers or immutable containers	N/A (no access to underlying infrastructure)
Infrastructure Provisioning	Hand-crafted	Hand-crafted, or potentially 'automated sysadmin' approach (CFEngine, Puppet etc)	Customised vendor-specific provisioning, which may only expose a deployment container such as application server	Provide 'automated sysadmin' including 'frying' of environments with tools such as Chef, Puppet or Salt	Highly automated, with environments typically 'baked' with tools such as Packer.i, Aminator or Docker	None required. Deployment artifact is typically uploaded via API / SDK to pre-provisioned platform
Application Configuration	Manual	Manual, potentially some automation via generated config files	Semi-automated	Semi-automated	Automated, typically provided via external service such as Zookeeper, Consul, etcd	Automated (at build time)
Inter-platform Service Discovery	None required	None required	Provided as part of the platform	Manual, or semi-automated (via config files)	Provided by external service within the platform (e.g. ZooKeeper, Consul, etcd)	Automated (at build time)
Routing	Typically none, as megaliths are typically only scaled vertically	Ingress traffic is typically hardware load-balanced to cluster of application instances. Internal communication typically via central load-balancer	Provided as part of the platform	Combination of software load-balancers (HAProxy etc) and P2P. MQ used for async communication	Typically P2P, or via service endpoints (via DNS etc). MQ broker provides routing for async or event-driven communicaiton	Typically events are sent to platform API or MQ broker, which handles routing
Observability / Monitoring	Single artifact to monitor. OS monitored at hardware level	Artifacts and instances monitored at cluster level, typically via tooling such as Nagios, Graphite and bespoke applications	Provided as part of the platform	Hybrid of monolithic and microservice monitoring	Services monitored individually by external centralised applications e.g. Logstash, Nagios, Reimann, Graphite	Platform monitoring via vendor PaaS console
Dev Tooling	Terminal for remote access to code, and (if lucky) IDE	IDE	IDE, potentially with vendor-specific plugins	IDE, external service virtualisation (enabling testing against QA servers), and scripted local service orchestration	IDE, service virtualisation (e.g. VCR, mountebank), local service orchestration (e.g. Ansible, Fig)	IDE (limited)

Microservice Maturity Model Proposal – Daniel Bryant (@danielbryantuk)

Name	Megalith Platform	Monolith Platform	Macro SOA Platform	Meso Application Platform	Microservice Platform	Nanoservice Platform
Ops Tooling	Shell scripts, quick wits and lots of coffee	Shell scripts, with some automated provisioning tools such as Puppet, Chef and Salt	Shell scripts and vendor-specific installs and tooling	Provisioning and config management, such as Chef, Puppet, and Ansible	Provisioning and config management with tooling such as Puppet or Ansible, and PaaS vendor-specific APIs / SDKs. Potentially tooling around Docker or another container-based OS (CoreOS, Project Atomic etc) ecosystem	None required
Testing	Witchcraft and voodoo	Manual testing, and (if lucky) unit tests and monolithic integratation test suite	Typically manual via QA and Staging environments. Some localised automated testing for components/services	Automated testing. QA and Staging environments allow manual validation	Extensive automated testing at component level. Automated canary deployments (capable of rollback) and synthetic monitoring in production	Synthetic monitoring in production
Iteration Cycle	Yearly	Quarterly - weekly (although companies such as Flickr, Google and Etsy have pioneered multiple daily releases)	Monthly - weekly	Monthly - daily	Within minutes	Within seconds
Delivery Model	Big bang	Big bang (some continuous integration)	Coordinated Big Bang via vendor tooling	Continuous integration, potentially with continuous delivery	Continuous delivery	Continuous delivery
Examples	Large 'enterprise' systems, such as insurance and financial software	Lots of companies (probably yours)	Typical modern 'enterprise' organisation, e.g. finance, insurance and travel	Lots of companies attempting to innovate (e.g Groupon, Sage)	Typical 'Cloud-native' or DevOps unicorn organisations e.g. Netflix, Amazon, Twitter	Amazon