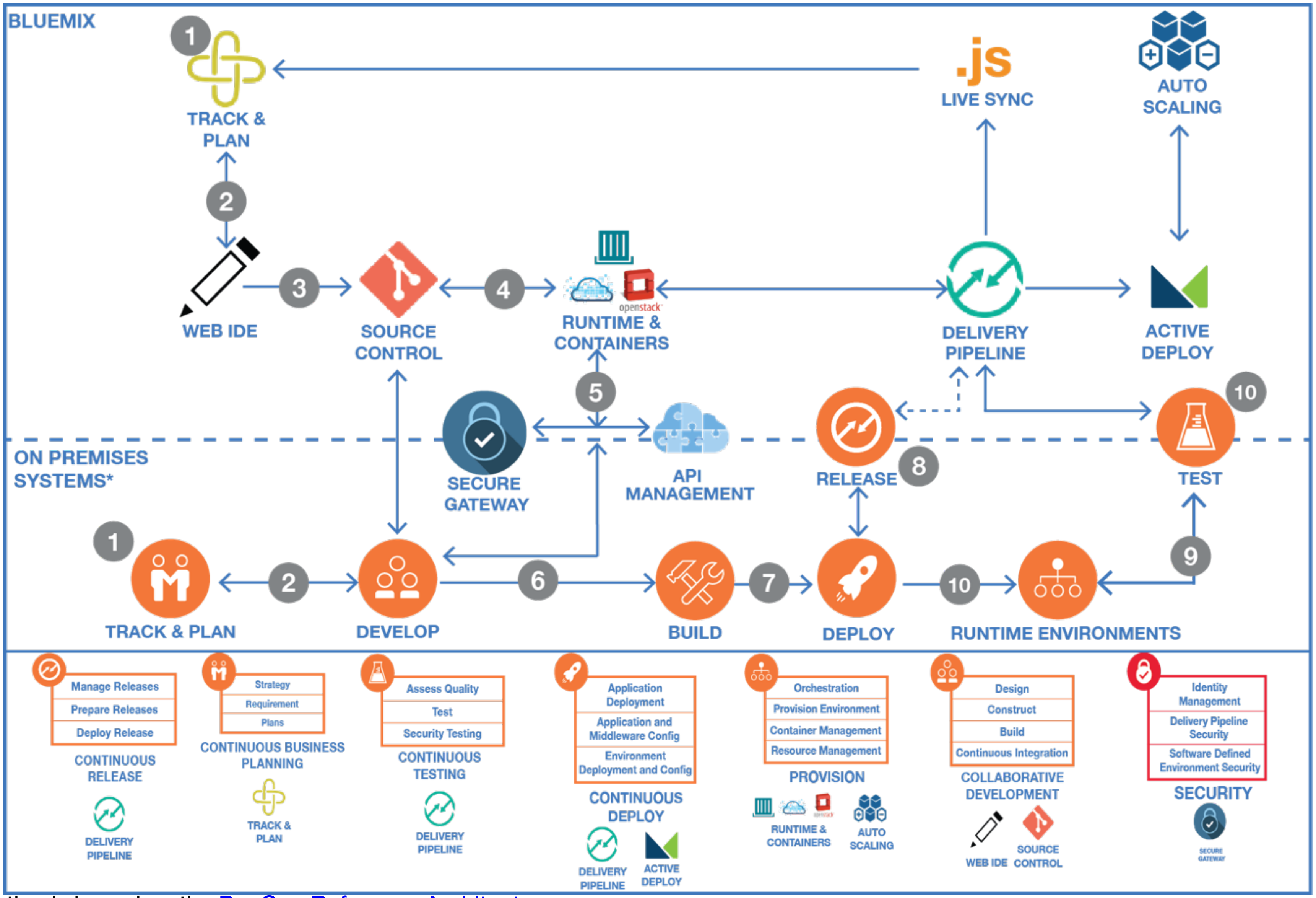
IBM **Cloud Architecture Center**

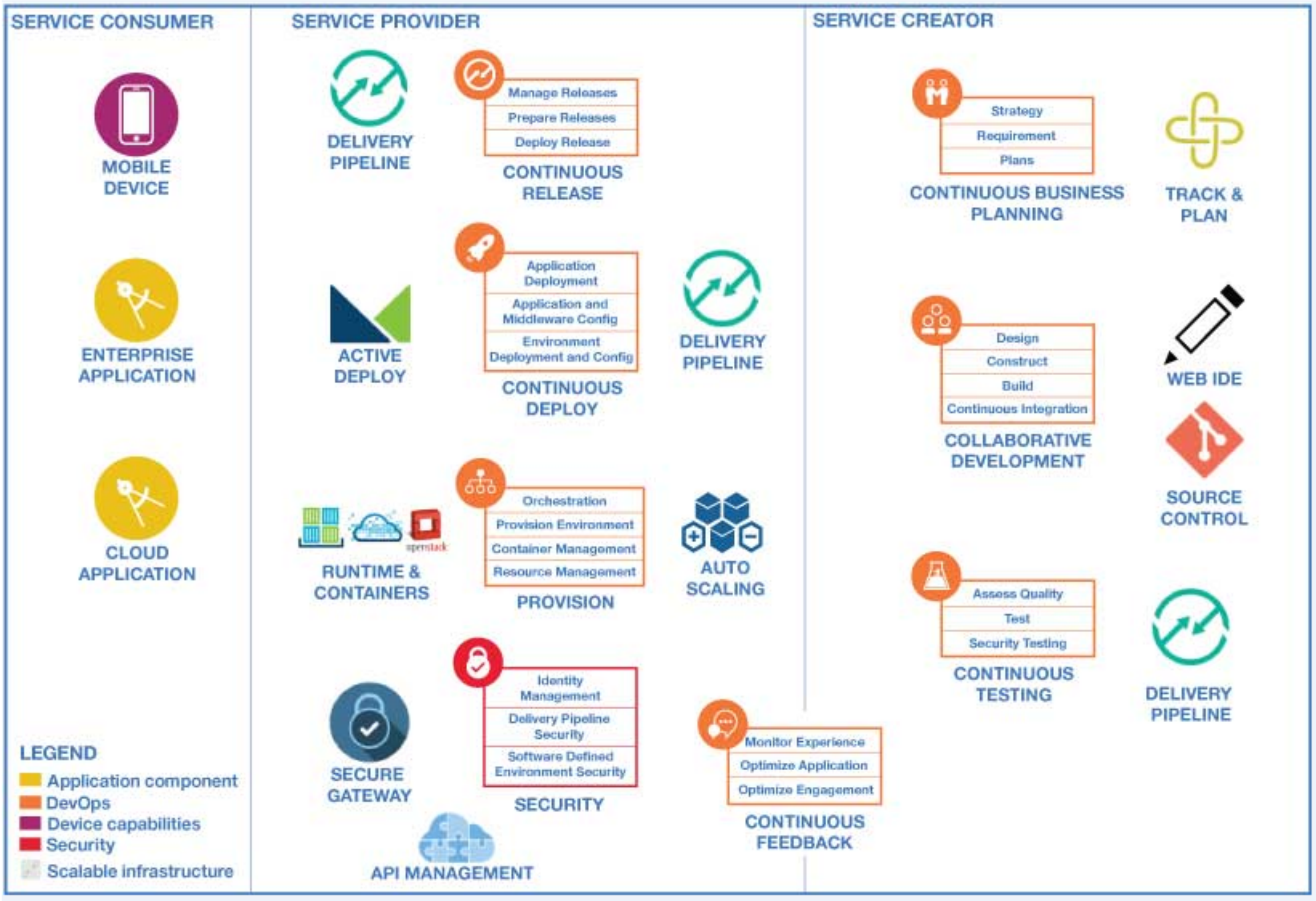
DevOps multi-speed IT

Cloud native applications can be developed and deployed significantly faster than cloud-ready applications or traditional systems of record applications. Close coordination is required when a composite system is deployed requiring changes to both sides of the delivery pipeline.



This solution is based on the DevOps Reference Architecture.

DevOps architecture overview – Bluemix mapping



Runtime flow

1. DevOps team members are enabled to access the continuous delivery platform. The line of business (LOB) leader and DevOps team designers collaborate to define an MVP candidate. The team lead establishes the team backlog and plan for a sprint and invites team members with IBM DevOps Services Track & Plan or the local tracking and planning system.

2. DevOps developers create code using their development tooling of choice or the integrated development environment (either fully web based or on-premises).

3. A DevOps team member links a new or existing Git repository to the team project with IBM DevOps Services Git integration, an existing on-premises source code management such as IBM Jazz SCM, or both.

4. Code is initially pushed from the IDE to the development environment for the initial application instance. This could be a container, a cloud foundry runtime, or an OpenStack virtual image that resides locally or in the cloud.

5. DevOps team security lead establishes a secure gateway to the backend core systems using the Bluemix® Secure Gateway service or API management which may reside locally or in the cloud.

6. A build engine manages source code, database scripts, property files, and build scripts in a version control system to maintain a complete history of every change made to the application. It ensures that when a library is changed, the dependent applications are rebuilt and dependency problems are connected for route cause analysis.

7. Release management tools help handle the growing number and complexity of releases, helping to plan, execute, and track a release through every stage of the delivery lifecycle. It results in reduced errors, while making large releases faster and more agile.

8. Test automation tooling can significantly reduce test cycle times, moving integration testing earlier in the development lifecycle and drastically improving application quality. Test scripts are automatically invoked as part of the deployment process.

9. Deployment engines orchestrate the deployment of builds to environments, configure the environment, trigger the automated tests, and return results. Stage gates are configured in the release management system that define when a build can progress between environments and be closer to final production deployment.

10. Delivery pipeline is represented as both a local implementation and a cloud-based deployment. Integration between the slower moving traditional development on local systems and the faster moving developments on cloud native systems happens within the delivery pipeline.

Components

COMPONENT	DEFINITION	IBM PRODUCT
Track and plan	Track work and team progress, create defects, see what's incoming, maintain project backlog, and plan work for future sprints.	IBM DevOps Services Track & Plan, Rational Collaborative Lifecycle Management
Integrated development environment	Web or locally based tooling developer consisting of a source code editor, intelligent code completion, build automation tools and a debugger.	Bluemix Web IDE, Eclipse based IDE
Runtime and containers	Environment where the code will be developed, tested, staged, and run.	Bluemix runtimes (Node.js, Liberty for Java, others), OpenStack virtual machines, IBM Containers
API gateway	Securely invoke and integrate code with APIs to and from other systems.	Bluemix Secure Gateway, Bluemix API Management, IBM API Management
Source control	Repository for sharing, storing source code, and versioning code drops.	Git with IBM DevOps Services, IBM Jazz SCM
Release	Help plan, execute, and track a complex release through every stage of the delivery lifecycle.	UrbanCode Release
Test	Integrated test suite to define test plans, test scripts, and manage test results. Automate testing of application components.	Rational Test Workbench
Delivery pipeline Build (on-premises) Deploy (on-premises)	Build, scan, test, integrate, and package apps before deploying. Automates builds and deployments to the cloud or local systems.	IBM DevOps Services Delivery Pipeline, UrbanCode Build, UrbanCode Deploy
Code synchronisation	Quickly update the application instance running on the Cloud and develop as you would on the desktop without redeploying. Ensures local and cloud based code stay synchronised.	Bluemix LiveSync

Business drivers

01

Line of business leaders quickly determine best alternatives for new business capabilities.

02

IT enables the business to experiment with entering new markets, introducing new products, and differentiating existing products.

03

Core systems development and delivery is streamlined and synchronized to enable better backend support for the innovation edge of the enterprise.

Requirements

Functional requirements

Coordination of application delivery

- **Close coordination is required when a composite system is deployed, requiring changes to both sides of the delivery pipeline. Cloud native applications can be developed and deployed significantly faster than cloud-ready applications or traditional systems of record applications. This allows the different speeds at which IT can react to change.**

Secure gateway and API management

- **Securely invoke and integrate code with APIs to and from other systems. Core system APIs are securely accessed through a secure gateway. APIs provided to external users are made available and managed through the API management services.**

Requirements

Non-functional requirements

Interoperability with core systems

- The continuous delivery tooling must support access to core system test environments and test data.
- Cloud native applications must interoperate with backend and core systems of record for transactions, data access, and synchronization.

Security

- The continuous delivery tooling must enable secure connections to backend systems.
- Cloud native applications must be designed, tested, and released to meet privacy, data security, and all corporate security compliance mandates.

Performance

- Creation of environments within the continuous delivery pipeline must complete in under 10 minutes.
- Developer collaboration, code sharing, and task tracking must be up to date across all team members in a matter of seconds

Usability

- The continuous delivery pipeline and tools must be very easy for team members to access and use.

Scalability

- The delivery pipeline must easily scale along with the enterprise plans for large numbers of DevOps teams working in parallel.
- Cloud native application infrastructure should support an elastic utilization model with an expected baseline of expected users and automatic scaling of resources to meet cyclic and peak demand.

Reliability

- The DevOps tooling, especially source code management, must reliably maintain baseline artifacts, allow for easy recreation from previous baselines, and allow for offline development for disconnected developer access.
- Cloud native applications must be designed, tested, and released to ensure reliable uptime and automatic failure detection and re-deployment of applications and containers without degradation of service.

For more information and for the latest IBM solutions and other assets, visit developer.ibm.com/architecture