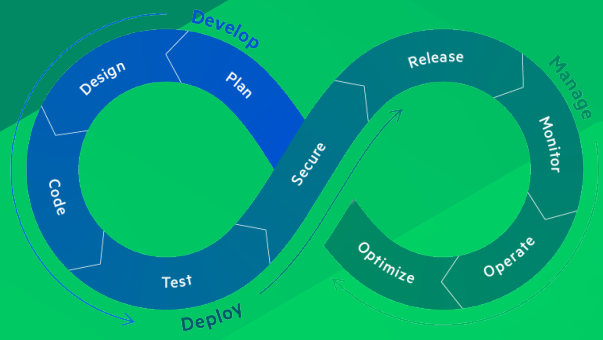
Progress Services

Continuous Integration & Continuous Delivery (CI/CD)

DATA SHEET



Progress Professional Services experts can quickly help you build a CI/CD strategy that is right for your business.

- Discovery Workshop
- Technical Process Assessment
- Strategy Discussion
- Recommended Approach and Tooling
- Introduction and usage of the OpenEdge DevOps Framework
- Implementation of the recommended approach

The prerequisites for using the OpenEdge DevOps Framework are:

- OpenEdge 12.2 or later
- Gradle 7.3.3
- Java 11
- pct, which is sued from DLC or pct, by default
- An ABL Project

The software delivery model in legacy platforms tends to be extremely complex and very labor-intensive. The separation of IT from development as well as monolithic, on-premises ecosystems dictate that software releases are done infrequently and at great risk. If that describes your organization, you need to build a repeatable process that includes integrated tooling, services, data and processes to enable your engineers to plan, build, test, release and manage your software delivery practice.

To compete in today's agile and frequently changing environments, releases need to occur in weeks, days or as needed with mature and well-organized DevOps strategies. Having DevOps best practices ensure efficiencies and extensibility as software platforms are expected to support microservices architecture, containers and cloud technologies.

To scale for continuous delivery and deployment, having a fully baked CI/CD strategy is required for DevOps teams to effectively deliver software releases quickly and reliably. A continuous and automated delivery methodology is the backbone that makes fast and reliable software delivery possible.

Introducing the OpenEdge DevOps Framework

As part of a typical application development lifecycle, mission critical business applications go through many cycles of incremental changes, complete with code changes, compilation, builds and validation. Often these cycles are repeated multiple times a day before a release is made available to users. Continuous integration (CI) is a practice that focuses on optimizing the process of generating these incremental builds and validating them. In the illustration below CI represents DevOps and describes a typical software development and delivery process.

For ABL applications, the OpenEdge DevOps Framework is designed to help with implementing an efficient CI pipeline that handles compilation, repository integration, testing and packaging. It also provides the convenience of sharing the CI pipeline configuration between the development and production build processes.

The OpenEdge DevOps Framework comprises a set of plugins designed to address the requirements of two types of users:

- Users who are new to the CI process and want a simple way to set up their pipeline.
- Advanced DevOps engineers with a complex CI process and additional flexibility needs.

The OpenEdge DevOps Framework provides the convenience of sharing the CI pipeline configuration between the development and production build processes.

Gradle Basics

Gradle is an open-source project automation tool, which is an evolution of the concepts used in Apache Ant and Apache Maven. Instead of traditional XML, Gradle uses a domain-specific language based on Groovy for project configuration. Gradle intelligently determines which parts of a build tree are up to date and executes only the needed parts. This process allows builds to be smarter, faster and more efficient because it eliminates redundancies in the build phase.

The OpenEdge DevOps Framework provides Gradle plugins to evolve the way ABL applications are built. The following basic Gradle terminologies and concepts are helpful when using the OpenEdge DevOps Framework:

- **Plugin** - Gradle plugins add task objects and conventions to your Gradle environment. They are the primary method of extending capabilities using Gradle.
- **Task** - A Gradle task is the smallest piece of work for a build. For example, compiling classes or generating ABL doc. It is the building block of Gradle functionality.
- **Task type** - Defines the blueprint of a task. Some tasks may require configurations like input or output parameters to provide certain capability. The blueprint for these parameters is defined by the task type for the piece of work that can be set while creating a task.
- **Task dependencies** - Gradle tasks can depend on other tasks.
- **Extension** - Plugins can add Extension, allowing you to configure several settings and properties. Task types use these settings and properties as global configurations to set some of their properties.
- **Groovy DSL**—You can write Gradle build scripts using a Groovy or Kotlin DSL.

Building an ABL project involves various steps such as managing project dependencies, compiling ABL files, performing unit tests, generating ABL API documentation, packaging the artifacts and publishing them to an artifact repository.