inQdo

WHITE PAPER.

Projen: The next step in project configuration

How automated project structures lead to more efficient development teams and more consistent codebases

March 2025

inQdo

CONTENTS.

Introduction

"Just setting up a new project." This seemingly simple task often grows into a complex challenge in modern development teams. This is not because starting a project itself is complicated but because the proliferation of different configurations, tools, and best practices makes keeping projects consistent increasingly difficult.

Project configuration is a term rarely heard outside technical circles, but it's a crucial component of every successful project within software development. It forms the foundation upon which software is built and maintained, and its quality directly impacts team efficiency and collaboration. This task often remains manageable in smaller teams with just a handful of projects. But what if you're working with dozens or even hundreds of projects that all need their configurations?

This isn't a hypothetical scenario for development teams working with more than 200 repositories—it's a daily reality. Each project within those repositories requires unique configuration files such as **package.json*, *tsconfig.json*, *.eslintrc.json*, *.prettierrc*, *.gitignore**, and CI/CD workflows. Adjusting these files often happens manually, which is time-consuming and increases the risk of human error. A small adjustment to linting rules means manually updating dozens or even hundreds of repositories. Implementing a new security best practice? That exercise takes days or even weeks.

This problem only grows as teams expand and projects become more complex.

This problem only grows as teams expand and projects become more complex. Inconsistent configurations can lead to:

- Time-consuming code reviews focusing on style and formatting issues rather than functionality
- Increased technical debt, as manual fixes are often applied ad hoc
- Frustration within teams, as the lack of standardisation complicates the onboarding of new team members

Traditional tools versus Projen

Traditional project configuration tools like Cookiecutter and GitHub templates have attempted to address these challenges. They often provide a good start by generating a project template, but that's where it ends. Once the project is running, it's up to the team to manually implement and track changes. CDK-init has this same limitation.

This leads to situations where configuration files quickly fall out of sync.

This leads to situations where configuration files quickly fall out of sync. A template that once contained all necessary components becomes irrelevant months later because best practices have changed. This makes maintaining consistency almost impossible.

This is where Projen comes in. Projen isn't just a tool; it's a philosophy. It treats configuration as code, meaning project rules and settings are centrally managed and automated. By creating a single source of truth in the form of a projenrc.ts file, Projen ensures consistent, up-to-date configurations in every project.

What distinguishes Projen from other tools is the combination of simplicity and power. It provides a streamlined way to start projects and a mechanism to manage and update them continuously. With Projen, you no longer need to worry about manual fixes or forgotten updates. Everything is automatically generated and managed based on the latest standards.

1

From CDK to Projen: A natural evolution

Understanding the power of Projen helps to look at a concept many development teams are already familiar with: AWS CDK (Cloud Development Kit). Both tools were conceived by the same developer: Elad Ben-Israel. Projen and AWS CDK use the 'construct' library, enabling a modular hierarchy. They also utilise JSII, allowing project templates to be used in other programming languages. This enables teams to work at a higher level of abstraction while technical details are handled automatically.

The parallel between CDK and Projen is striking. Both tools abstract complex processes and work with a modular hierarchy that is adaptable and scalable. For CDK, these are constructs that lead to CloudFormation resources; for Projen, they are components that generate files.

AWS-CDK

AWS CDK works with a clear hierarchy:

- App forms the base and contains one or more stacks
- Stack contains one or more constructs
- Constructs ultimately lead to concrete CloudFormation resources

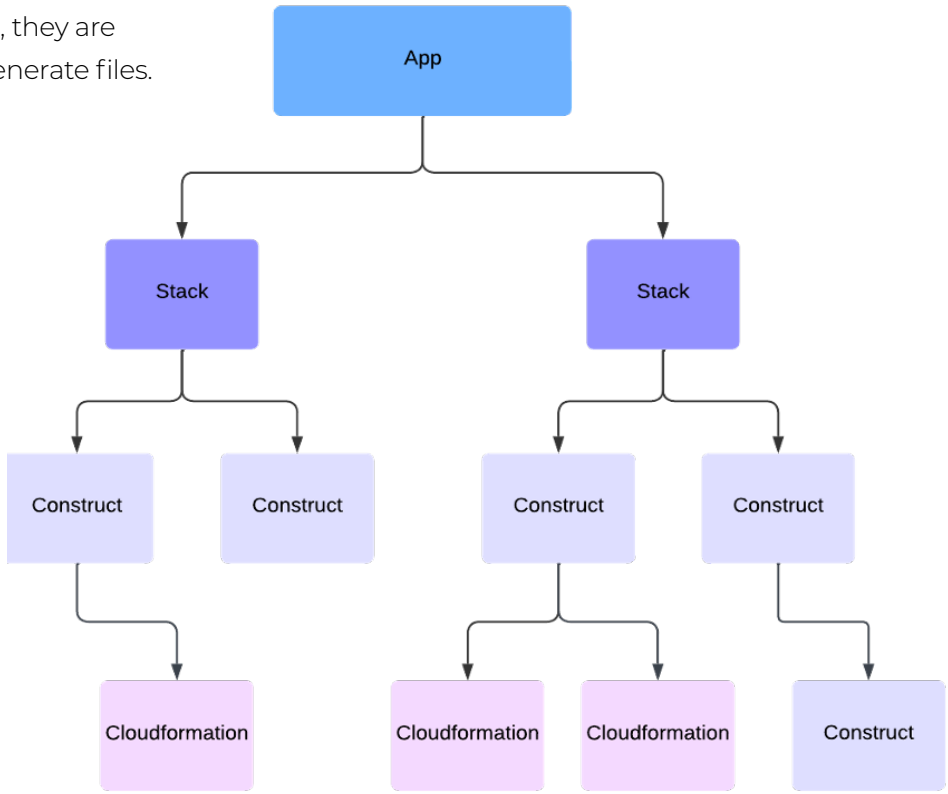
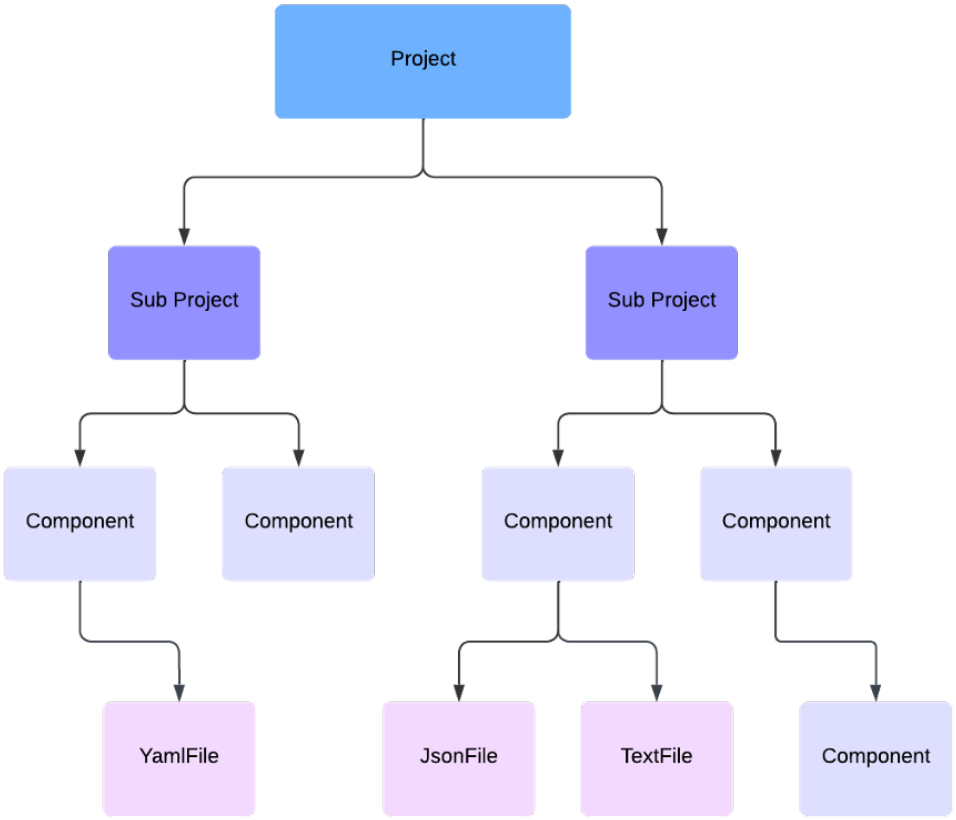


Figure 2.1 > Schematic representation of AWS CDK



< Figure 2.2
Schematic representation of Projen

Projen

Projen follows a similar concept:

- Project forms the base and can contain sub-projects
- Sub-projects have components
- Components ultimately generate concrete configuration files

This parallel is no coincidence. Just as CDK frees developers from manually writing CloudFormation templates, Projen frees teams from manually managing configuration files. It elevates project configuration to a higher level of abstraction, where it's easier to manage and maintain. With each project started using Projen, a central projenrc.ts file is generated. This defines the complete project configuration, from dependencies and linting rules to build scripts and workflows.

The practical impact

In practice, this approach directly solves several persistent problems that many development teams experience:

- **Consistency across projects**
  Managing more than 200 repositories means that even small changes, such as adjusting a linting rule, can have an enormous impact. By treating configuration as code, such updates become a matter of minutes rather than days or weeks.

- **Simpler collaboration**
  Code reviews become more effective

because teams can concentrate on functionality rather than formatting or configuration. A pull request shows only the relevant changes, not the noise of inconsistent configurations.

- **Faster onboarding**
  New team members, or developers temporarily joining a project, can get started immediately. With one command, they have a fully configured development environment that meets all team standards:

< Figure 2.3
Difference in command prompt between CDK and Projen

- **Automated security**
  Security tools like cdk-nag can be integrated by default. This ensures that security guidelines are consistently applied across all projects.

2

Projen in practice

Projen is more than a theoretical concept; it's a tool that has proven to streamline processes, save time, and improve work quality. It becomes clear how teams, such as inQdo, use it to solve daily challenges.

One of Projen's greatest strengths is its ability to generate and maintain configuration files automatically. Where developers traditionally spend considerable time manually creating and managing project configurations, Projen offers an automated solution that saves time and guarantees consistency.

Take, for example, setting up a CDK project. Without Projen, a developer must manually create and configure a series of files. This includes essential files like .gitignore, package.json, and tsconfig.json, but also more complex components such as CI/CD workflows for GitHub Actions. The process requires careful attention, as a small error can lead to inconsistent configurations or implementation problems.

With Projen, this complex task is reduced to a single command. By simply executing:

projen new awscdk-app-ts

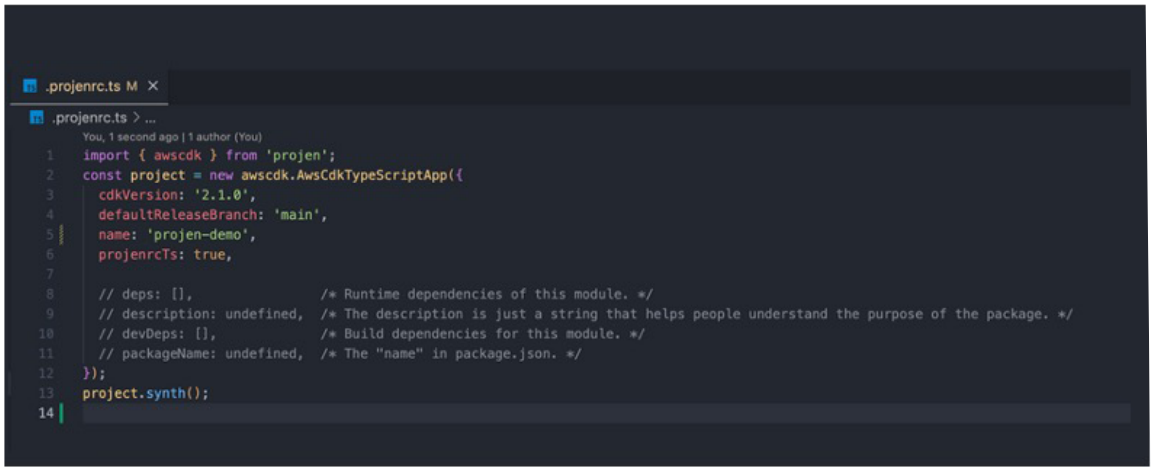
a fully configured project environment is automatically generated. Within seconds, the developer has a project structure that meets the team's predefined standards.

The generated project structure looks like this:

my-project/
- ***.gitignore***
- ***package.json***
- ***tsconfig.json***
- ***.eslintrc.json***
- ***.projen/***
- ***workflows/***
- ***src/***

Each file is automatically aligned with the set standards, from linting and formatting rules to workflow configurations. What was previously a time-consuming and error-prone process has become an efficient, streamlined step in development teams' daily practice.

Projenrc.ts



< Figure 3.1
Central configuration within Projen: Projenrc.ts

Projen gives development teams back time and provides a solid foundation to build upon. By managing configurations centrally and keeping them automatically up-to-date, developers gain the freedom to focus on what truly matters: building innovative solutions.

Managed versus unmanaged files

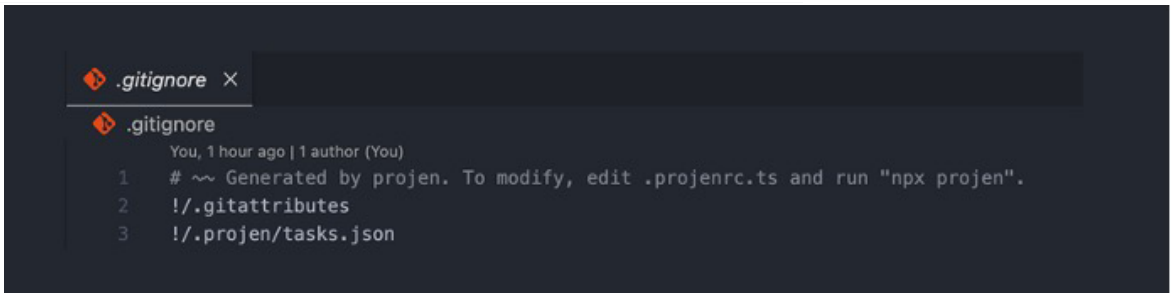
What truly distinguishes Projen from other solutions is the concept of 'managed' and 'unmanaged' files. This fundamental principle determines how configurations are managed and kept up-to-date.

Managed files: Automatic and consistent

Managed files are fully managed by Projen and regenerated with each update. These include configuration files for:

- Linting and code formatting
- Build processes
- CI/CD workflows
- Dependency management

Managed files can be identified by the Projen marker



< Figure 3.2
Managed files are managed by Projen

These files are identifiable by a special Projen marking indicating they are automatically managed. Manual changes to these files are overwritten—a deliberate choice that guarantees consistency.

Unmanaged files:
Flexibility where needed

Not all files need strict control. Unmanaged files are generated once as a starting point and can then be freely modified. This is ideal for:

- Example code and templates
- Project-specific implementations
- Documentation
- Custom configurations

Example:
Suppose you have a *.gitignore* file containing rules for ignoring files that are pushed to Version Control. Projen ensures these rules always stay up-to-date according to team standards. Manual additions to the file are ignored, but these should be added via the projenrc.ts instead.

3

Configuring configuration

What makes Projen different?

The fundamental limitation of traditional solutions is that they treat configuration management as a one-time action—setting up a project—rather than a continuous process. In practice, a development team constantly evolves. New security patches must be applied, coding standards must be tightened, and dependencies need updating. Without automated management, this becomes an almost impossible task.

> The difference with Projen becomes clear when we look at how teams handle, for example, a new security best practice. With traditional solutions, this would mean:
>
> 1. Updating the template or script for new projects
> 2. Making an inventory of all existing projects that need modification
> 3. Manually going through all repositories to implement the change
> 4. Conducting code reviews on each change
> 5. Hoping no project has been overlooked

With Projen, however, the new practice is added to the central configuration. Because it's Configuration-as-Code, the configuration of the configuration becomes manageable in version control such as Git/GitHub. This creates a history of changes and an overview of who modified what.

Moreover, the project template is made available as a 'package'. When starting a new project, the package can be downloaded, and the new project depends on this package. This means that when the package content changes (the configuration), the project is automatically updated to the latest version of the project template by updating the package. This happens with one command (npm update), which implements all changes. The execution of package updates can also be automated.

Projen changes are automatically and consistently implemented, without manual work and with minimal risk of errors. It goes beyond generating an initial setup and offers a continuous management model.

Traditional tools	Projen
Focuses on initial project setup	Manages projects throughout their entire lifecycle
Manual updates for configurations	Automatic updates via managed files
Limited to one-time templates	Use of reusable components
No integrated dependency management	Complete dependency management within projenrc.ts

4

From theory to practice: Implementing Projen

The move to Projen requires a thoughtful approach. The reality of a development team with more than 200 repositories is that you can't change everything at once. Ingrained working methods, existing configurations, and ongoing projects require a gradual transition.

A good example of this approach is how an inQdo team experimented with Projen in a new serverless project. This project was small enough to oversee but complex enough to test Projen's capabilities. The initial results were promising: where the team normally spent hours setting up and configuring a development environment, they now had a fully configured project within minutes.

But the real value became clear when an update to the linting rules was needed. This situation regularly occurs in practice: a team discovers a better way to structure code and wants to implement this standard across all projects. Without Projen, this would mean manually adjusting each project. With Projen, it was a matter of modifying the rule in the central configuration file and letting the change propagate automatically.

Benefits of Projen

This experience led to a broader insight: Projen isn't just a tool for project configuration; it's a way to democratise development standards. Treating configuration as code makes it possible to build best practices directly into the project structure.

For instance, new team members no longer need to wade through extensive documentation to understand how a project should be set up—the standards are baked into the configuration. Previously, it took days to get a new developer fully productive. They had to delve into various configuration files, set up local development environments, and understand project-specific settings. Now it's a matter of executing one command to have a fully configured development environment that meets all team standards.

Another example is the integration of security tools. By including CDK-nag as standard in the Projen configuration, security checks become a natural part of the development process. The tool cannot simply be disabled or bypassed, ensuring consistent security standards across all projects. However, even within existing projects, Projen ensures that security can be implemented more easily. Take, for example, a recent project

where a critical security update needed to be implemented across all repositories. Where this would previously have taken an entire sprint, it now became a streamlined operation completed within a day. Not only was this more efficient, but it also eliminated the risk of accidentally skipping a repository.

Practice also shows that teams need different levels of automation. Some projects require strict control over every configuration setting, while others need more flexibility. Projen supports this through the distinction between managed and unmanaged files. A good example of this is the treatment of GitHub workflows: the basic workflows for testing and deployment are automatically generated and updated, while teams retain the freedom to add project-specific workflows.

Configuration management automation has had an unexpected but positive effect on code review efficiency. In traditional development environments, inconsistent configurations mean that important code changes are masked by superficial formatting differences. It's comparable to reviewing a document where not only the content has been modified, but the formatting has also been randomly changed—it becomes almost impossible to identify the actual substantive changes. Projen eliminates this problem by enforcing consistent formatting. This makes functional changes, such as new features or security patches, immediately visible in code reviews. Teams can concentrate on what truly matters: the logic and functionality of the code. This increased visibility of relevant changes speeds up the review process and improves the quality of the reviews themselves.

Common pitfalls and how to avoid them

1. Relying on automation without understanding
Challenge: Team members fully trust Projen without understanding how it works.
Solution: Combine automation with training and documentation so team members understand what Projen does and how they can adapt it.

2. Overwhelming complexity at the start
Challenge: Implementing too many standards at once can be confusing.
Solution: Start with a simple configuration and add complexity gradually based on feedback.

3. Insufficient team communication
Challenge: Not everyone in the team understands the benefits of Projen.
Solution: Regularly discuss the impact of Projen and share successes, such as time savings and improved consistency.

5

Looking into the future

Software development is in constant evolution. Teams face increasing complexity, rapid technological changes, and growing demand for standardised processes. Once a simple task, project configuration has become a strategic challenge.

Trends driving this change:

- **Scalability:** The number of repositories and projects within organisations is increasing, and consistency is needed.
- **Automation:** Tools such as CI/CD and AI support developers but require standardised configurations to perform optimally.
- **Team diversity:** Development teams increasingly consist of people with varying technical expertise, increasing the need for user-friendly tools.

Projen plays a key role in this landscape by treating configuration as code and solving a fundamental problem in software development.

Automation is a core component of modern software development. Developers rely on a wide range of automated tools, from generating builds to executing tests and deployments. Projen seamlessly aligns with this trend by automating configurations. With the rise of AI tools such as GitHub Copilot and other code assistants, having clear and consistent configurations becomes increasingly important.

An interesting development is Projen's integration with modern development tools. Treating configuration as code makes it possible to use the same tools for configuration management as for software development. This means, for example, that teams can benefit from IntelliSense and type-checking when modifying project configurations—a functionality that was previously only available for application code.

The next step in modern software development

The challenges of project configuration aren't unique. Every growing development team recognises the struggle with inconsistent configurations, time-consuming updates, and difficult onboarding of new team members. What is unique is how Projen tackles these problems. Treating project configuration as code transforms a traditional pain point into a strategic advantage.

Practice shows that the impact goes beyond just efficiency. Teams experience a fundamental shift in how they think about project configuration. It's no longer a necessary evil but an integral part of the development workflow. This mindset shift leads to better code, faster development cycles, and more focus on what truly matters: building valuable software

6

Steps to get started

For teams considering the switch to Projen, a phased approach is essential. Start small, with one project where the team has room to experiment. This can be a new or existing project due for a refresh. The most important thing is that the team has the freedom to learn and adapt.

During this phase, focus particularly on:

- Defining team standards that are truly valuable
- Identifying configurations that benefit most from automation
- Gathering feedback from team members about what works and what doesn't

You can expand to other projects once the first project is successfully running with Projen. Experience shows that teams are often so convinced of the benefits at this point that they themselves ask to migrate more projects.

An investment in the future

The future of software development lies in automation and standardisation. Tools like Projen play a crucial role in this. By treating project configuration as a first-class citizen in the development process, teams lay a solid foundation for future growth and innovation.

The integration with modern development tools and the ability to manage configurations programmatically prepare Projen for the next wave of innovations in software development. Whether it's AI assistants, new security requirements, or yet unknown technological developments—a structured, code-first approach to project configuration ensures teams are ready for whatever the future brings.

Start today!
Want to know more about how Projen can improve your development processes?

Contact our experts:
info@inqdo.com
+31 85 2011161
inQdo.com

Start simplifying your project configuration today and experience the power of Projen for yourself!

7

Key lessons from this whitepaper

1. Why Projen?

Projen solves fundamental problems that other tools like Cookiecutter and GitHub Templates don't address:

- It provides continuous management rather than just initial setups
- It eliminates manual updates by managing configurations centrally
- It increases productivity and reduces errors through standardisation

2. How does Projen work?

Projen uses a centralised configuration file (projenrc.ts) to ensure consistency. This file acts as the source of truth for each project and automates:

- The creation and updating of files such as .gitignore, package.json, and CI/CD workflows
- The integration of security tools like cdk-nag
- The implementation of team standards for linting and formatting

3. Practical experiences with Projen

At organisations like inQdo, Projen has proven valuable by:

- Saving time when updating configurations
- Accelerating the onboarding of new team members
- Improving collaboration within teams by maintaining consistent standards

8

Recommended resources for further study

Official Projen documentation:
projen.io
Find detailed guides and examples here.

GitHub repository:
github.com/projen/projen
View the codebase, open issues, and latest releases.

Community discussions:
Join forums and communities to exchange experiences and discover best practices.

About inQdo

inQdo is an AWS Advanced Consulting Partner specialising in cloud solutions and digital transformation. With a team of experienced cloud developers, we help organisations implement innovative solutions that create business value.

info@inqdo.com
+31 85 2011161

---

inQdo
Coltbaan 1-19
3439 NG Nieuwegein