DevOps made easy with GenAI

In this post, I will tell you about a winning hackathon project at Dell global hackathon. The idea is to use **Gen-AI** to help **lower the barrier to entry** and simplify the creation of infrastructure as code templates. By leveraging Gen-AI, we aim to streamline the development workflow and enhance the overall efficiency of managing complex infrastructure.

Infrastructure As Code

Think about Infrastructure as Code (IaC) as having a magic book that tells computers how to build themselves, making things quicker, easier, and less error-prone for people who work with computer systems.

DevOps automation plays a crucial role in simplifying the management of cloud infrastructure. By utilizing **Infrastructure as Code**, we transform infrastructure resources into software assets. This approach allows us to leverage software tools for automating tasks such as provisioning, configuration, and ongoing maintenance of these resources.

The challenge with IaC

It's essential to acknowledge that writing IaC is still a **challenging endeavor**. It demands a rare combination of both **development skills** and **operational expertise**. Additionally, IaC involves interacting with live systems, which can be rather **complex** to debug and troubleshoot due to their distributed and asynchronous nature.

This complexity is amplified when dealing with **edge infrastructure**, which, by definition, is more distributed and heterogeneous. As a result, the development process becomes significantly more intricate.

Using Gen-AI to reduce the barrier to entry of developing IaC

One of the key benefits of using natural language to generate code is the ability to reduce significantly the learning curve that is often associated with code development in general and devops automation specifically.

Indeed according to a recent research by McKinsey — AI Unleash: Generative AI is poised to transform software development in a way that no other tooling or process improvement has done… As the technology evolves and is seamlessly integrated within tools across the software development life cycle, it is expected to further improve the speed and even quality of the development process.

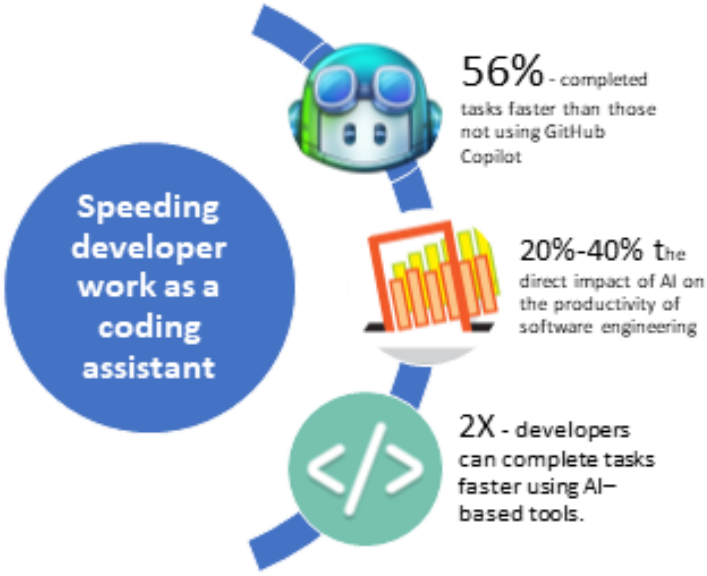


Figure 1 The benefits of using Gen-AI as coding assistant a.k.a co-pilot. (Source: McKinsey)

The challenges with out of the box Gen-AI tools

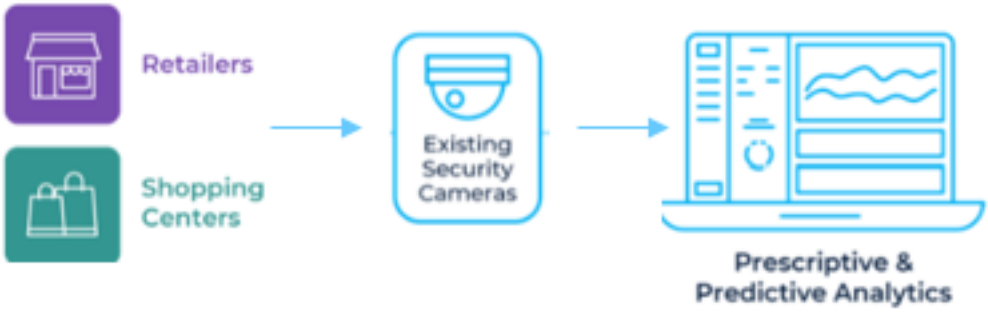
Handling custom development patterns: The challenges of using the out-box gen-ai tools such as GitHub co-pilot is that it was pre-trained on generic language and development patterns therefore requires special fine-tuning to address more special languages and development patterns.

To address those challenges lets start by examining a real life use case and follow the steps that a typical DevOps engineers of a software company need to go through to develop the IaC code for automating the provisioning and configuration of his software product.

We will then explore the steps that we took to solve this challenge of generating IaC templates that will be fit this specific use case.

Exploring Edge Orchestration Through a Customer Scenario

Let's delve into the realm of edge orchestration with a practical example involving one of Dell's clients. Picture a video analytics company that specializes in analyzing footage from cameras installed in retail environments, such as supermarkets and shopping centers. Their goal is to enhance the sales performance of their clients.

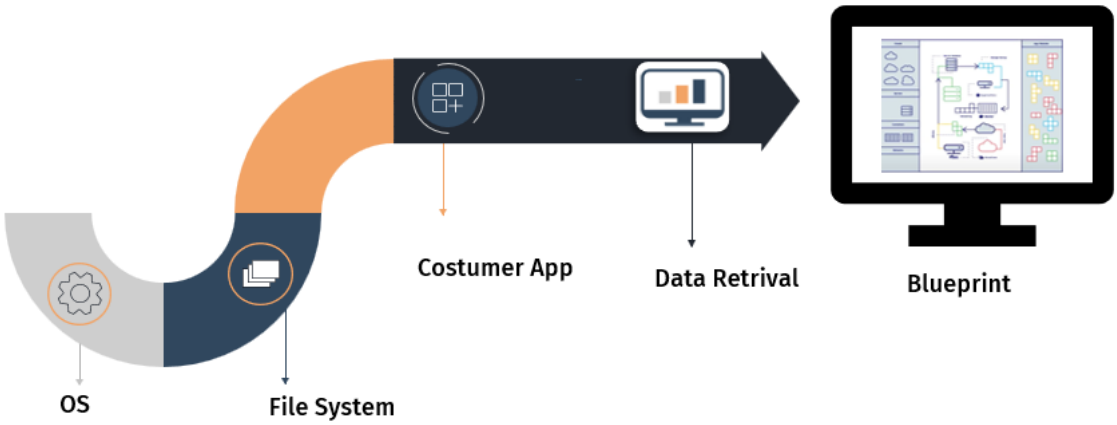


This customer faced the challenge of deploying their software application across 5,000 cameras, serving as edge devices in various shopping centers. This complex task involved several steps like:

1. Selecting the appropriate Operating System (OS) for installation.
2. Establishing a file system.
3. Implementing their software onto these 5,000 cameras.
4. Configuring the network to connect these cameras with their application.
5. Scheduling an automatic shutdown of the cameras between 10:00 PM to 6:00 AM.

6. Planning for periodic software updates.

Each of these tasks represents a component of a configuration file. This file acts as a detailed blueprint, outlining the necessary environment and services to be set up.



For a demonstration tailored to this customer, Dell typically assigns a solution engineer to craft this blueprint.

This process often spans several weeks, encompassing diligent effort and occasionally, moments of frustration, as the engineer acquaints themselves with the unique installation requirements and formulates the precise blueprint. Without the expertise of a Dell solution engineer, the customer might spend months developing this solution on their own — a timeframe that is frequently a deal-breaker in this fast-paced industry.

So how can we address these challenges?

You guessed it — we're using GenAI 😊

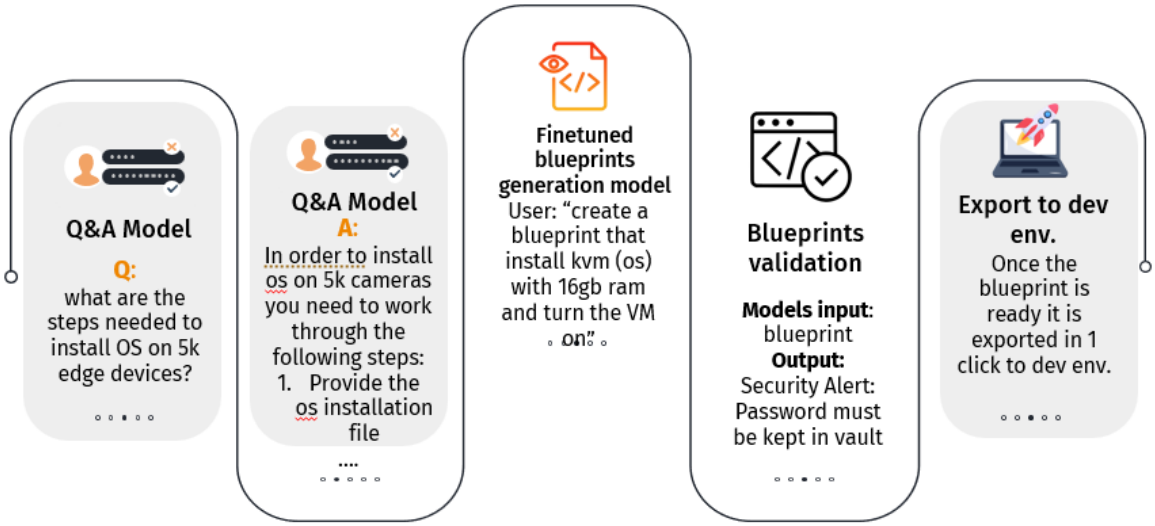
Let’s briefly overview our solution before diving deeper. We have two main models in our solution:

1. A Q&A model that helps answer questions about Dell’s documentation and internal technical resources.
2. A Code Autocompletion model that assists in creating blueprints.

There’s also a Validation model in development, but we’ll skip the details for now. Additionally, our solution smoothly integrates into a development environment for testing purposes.

Now, let’s revisit our Dell video-analytics customer scenario to see how our tool can simplify and enhance their process. Instead of spending weeks with a Dell engineer, our customer’s engineer, whom we’ll call Nelly, uses our Q&A model.

She asks it, “What are the steps to set up an OS on 5,000 cameras?” and gets the necessary steps in response. Next, Nelly starts writing the actual blueprint with the help of our second model. Once her blueprint is complete and validated, she can upload it to a development environment to test it on real devices. More on this in a bit.



Data and Modeling Insights

What kind of data and models are we leveraging for this project? First, let's discuss what we didn't use.

You might be wondering, "Why not just use Copilot or a commercial GPT for queries right off the bat?" We gave that a shot, but it fell short for our specific need to generate configuration files. This was mainly because our proprietary internal data wasn't familiar to the model, leading us to the necessity of fine-tuning.

Our modeling Strategy:

For the Q&A model, we've implemented a straightforward RAG architecture with a private GPT-3.5 instance. The data we're utilizing comes from both internal and external sources, including Confluence and data on dell.com. Interested in RAG? There's some excellent material out there for a deeper dive.

Creating Blueprint/Configuration Files

Our focus is on two primary objectives:

1. Developing a code completion model that seamlessly integrates with your IDE.
2. Crafting a model capable of generating blueprints from descriptions.

Code Autocompletion Model:

For the fine-tuning of our code autocompletion feature, we chose Startcoder. This model boasts 15 billion parameters and has been trained using a rich dataset from GitHub, encompassing over 80 programming languages, commits, issues, and Jupyter notebooks. As of May 2023, Startcoder has surpassed other large language models (LLMs) in popular programming benchmarks, featuring a context length of 8K — the largest at that time.

Startcoder's capabilities are diverse, ranging from acting as a technical assistant and offering code autocompletion to modifying code based on instructions and explaining code snippets in plain language.

Data Preparation for code autocompletion

In preparation for fine-tuning with LoRA, we needed several thousand YAML configuration files, or blueprints, for training. Facing a shortage of these files, we employed augmentation techniques. This involved rearranging the order of various fields

within the files, akin to shuffling the order of keys in a dictionary. Additionally, we tweaked the files by adding or removing certain optional fields to further diversify our dataset.

Our fine-tuning process with LoRA utilized 30,000 samples. It required some adjustments in parameters, like context size and LoRA rank, among others, to achieve our target outcome. We conducted manual evaluations of the model, with our engineers testing it across various use cases.

Serving the model and integration with IDE:

For the deployment of our model, we adopted the Text Generation Interface (TGI) provided by Hugging Face. This interface was particularly effective due to its compatibility with the llm-vscode extension from Hugging Face, facilitating a smooth integration with the development environment.

During our assessment phase, engineers attempted to create blueprints using the model's assistance. This process was crucial in identifying the scenarios where the model excelled and pinpointing the types of blueprints that posed challenges for it. Through iterative refinements of both the data and the model's parameters, we progressively enhanced its performance. This continued until we reached a point where our test engineers expressed satisfaction with the model's capabilities.

Generating Blueprints from Descriptive Language

Model Overview

To refine our model for interpreting instructions, we utilized Code Llama. Code Llama belongs to a series of advanced language models for coding, built upon the Llama 2 architecture, known for its leading-edge performance among accessible models. Specifically, we employed the 13-billion-parameter version of Code Llama for instruction-based fine-tuning.

Few-Shots Inference Approach

In the process of generating blueprints from user descriptions, we initially experimented with a few-shots prompting technique. This approach involves providing the model with a set of examples in its input, followed by a request for the model to generate a new blueprint based on these examples.

description 1:
please write a cloudify blueprint that creates infrastructure on AWS using Terraform.
the main.tf file will be provide via a link to a zip file and a path inside the zip.

cloudify blueprint 1:

BLUEPRINT 1 example

##

description 2:
please write a cloudify blueprint that creates VPC AWS service using the clouidfy terraform plugin.

cloudify blueprint 2:

BLUEPRINT 2 example
##

description 3: this time this is the user query:

please write a cloudify blueprint that creates S3 AWS service using the clouidfy terraform plugin
the user should provide the source location and source path as inputs.

cloudify blueprint 3: model response here

This approach proved effective only in specific scenarios and didn't offer the broad applicability we needed. Therefore, opting for fine-tuning became the logical next step.

Instruction fine-tuning

Data Preparation

To effectively fine-tune the model with instructions, it's essential to prepare sample pairs in the following format:

- Instruction: A clear, natural language description specifying the desired configuration file.
- Answer: The corresponding correct configuration file, or Blueprint.

Here’s an example to illustrate:

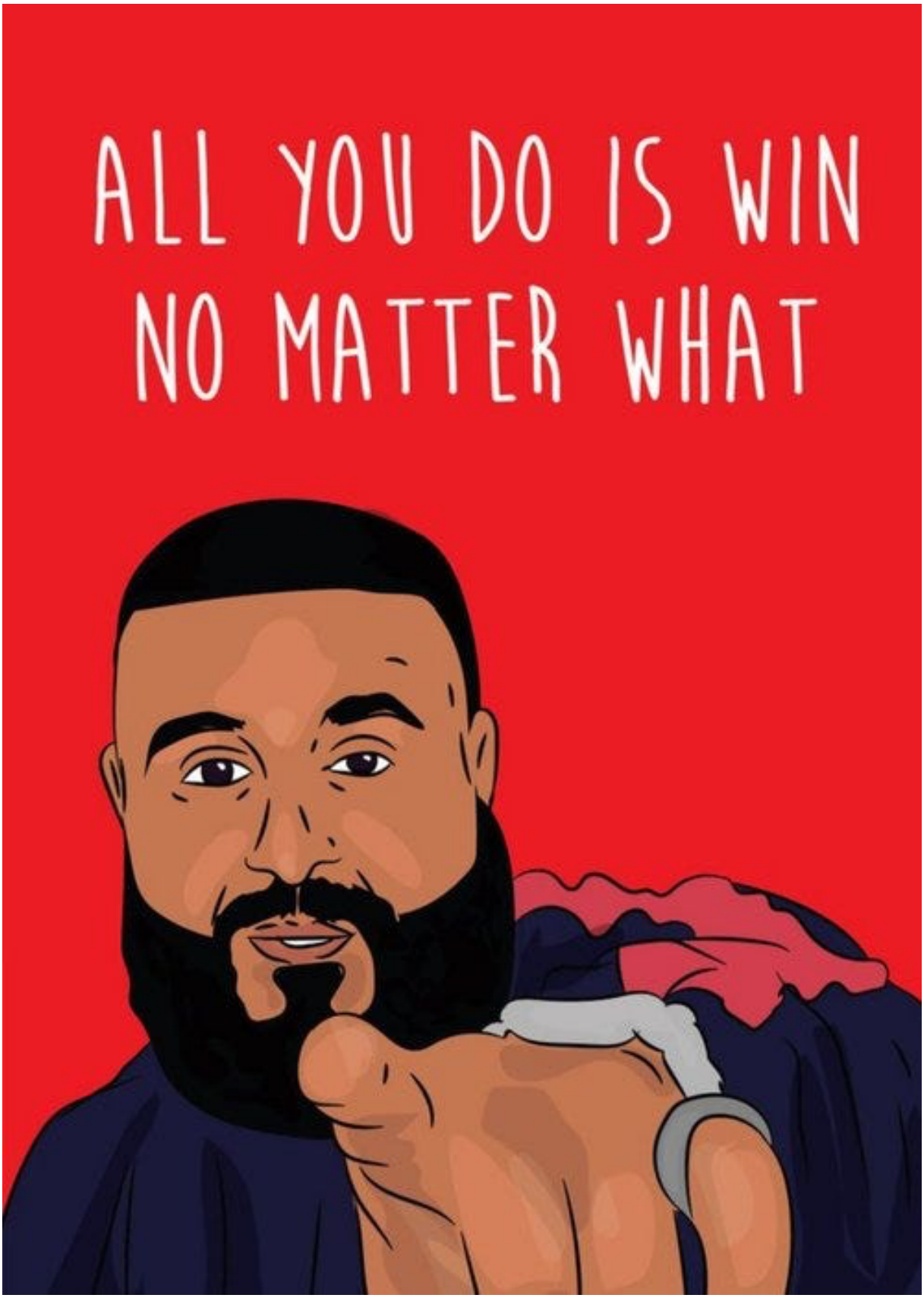
Please write a blueprint that deploy s3 on aws	tosca_definitions_version: cloudify_dsl_1_5
	
	description: This blueprint deploys some AWS infrastructure using Terraform.
	
	imports:
	  - cloudify/types/types.yaml
	  - plugin:cloudify-terraform-plugin?version= >=0.15.0
	  - https://raw.githubusercontent.com/cloudify-community/eaas-example/master/utils/custom_types.yaml
	
	labels:
	  csys-obj-type:
	  values:
	  - service
	
	inputs:
	  resource_prefix:
	  display_label: Resource Prefix
	  type: string
	  description: |
	  If not provided, it will be auto-generated.
	  default: ''
	  constraints:
	  - pattern: '(^ *$)\|(^[a-zA-Z][a-zA-Z0-9]+$)'
	  region_name:
	  display_label: Region Name
	  type: string
	  default: 'us-west-1'
	
	node_templates:
	  prefix:
	  type: eaas.nodes.UniquePrefixGenerator
	  properties:
	  predefined_value: { get_input: resource_prefix }
	
	  terraform:
	  type: cloudify.nodes.terraform
	  properties:
	  resource_config:
	  installation_source: https://releases.hashicorp.com/terraform/0.14.3/terraform_0.14.3_linux_amd64.zip
	
	  terraform_module:
	  type: cloudify.nodes.terraform.Module
	  properties:
	  resource_config:
	  environment_variables:
	  AWS_ACCESS_KEY_ID: { get_secret: aws_access_key_id }
	  AWS_SECRET_ACCESS_KEY: { get_secret: aws_secret_access_key }
	  variables:
	  bucket_name: { concat: [{ get_attribute: [prefix, value] }, bucket] }
	  bucket_region: { get_input: region_name }
	  source:
	  location: s3-templates/tf-bucket-master.zip
	  relationships:
	  - target: prefix

Creating the descriptions with gpt3.5

To compile our dataset, it was necessary to generate descriptions for each blueprint. For this, we utilized GPT-3.5 to create descriptions corresponding to each file. We experimented with various prompts to produce effective descriptions. Once we had our descriptions, we proceeded with the data preparation for modeling as previously outlined.

This development involved numerous iterations and thorough evaluations, as mentioned earlier. The goal was to refine the process until users of the tool genuinely felt its benefits, and we continually strive for further improvements.

Winning the Global Dell Technologies Hackathon


ALL YOU DO IS WIN
NO MATTER WHAT

The Dell ISG (Infrastructure Solutions Group) Global Hackathon is a yearly event, bringing together teams worldwide to address critical business, societal, or environmental challenges through innovative technology solutions. I had the privilege of leading a talented Israeli team, comprising brilliant engineers and data scientists. We joined forces in this hackathon, where over **2,200 team members** globally collaborated to present **624 remarkable projects.**

Despite facing a difficult situation in Israel, our team showed incredible teamwork and commitment. And it all came to fruition — we won the Hackathon! This victory isn't just a personal achievement; it's proof of the amazing things that can happen when people work together, innovate, and never stop striving for the best. Now, we're focused on growing our solution and turning it into a product for Dell's customers worldwide.

Thank you for following our journey thus far. If you're curious to learn more about our project or have any questions, feel free to reach out.