Building a Cloud-Native Toolset for Flexible, Continuous, Automated Simulation-Based Testing

Jeremy Loomis, Alex Matthews
NextGen Federal Systems
Morgantown, WV
jloomis@nextgenfed.com, amatthews@nextgenfed.com

ABSTRACT

The discipline of software testing is changing to align with the automated processes of Agile DevSecOps. In a DevSecOps environment, automated testing is executed by running test scripts or scenarios against the System Under Test (SUT) without human intervention. With many types of software (such as military applications), a challenging aspect of this testing is generating synthetic data and activity to stimulate the SUT.

Modeling and simulation (M&S) can help by providing representative test data from synthetic actors in a synthetic environment. Building on current M&S tools, a Simulation-Based Testing (SBT) approach can inject synthetic test data into a Continuous Integration / Continuous Deployment (CI/CD) pipeline. SBT requires two interconnected elements: M&S-as-a-Service (MSaaS) services and an Automation & Orchestration ecosystem.

This paper describes our Simulation And DevOps Integration Environment (SADIE) project, which built a proof-of-concept SBT toolset using existing COTS/GOTS software. SADIE is a cloud-native (deployable to any Kubernetes cluster) modular toolset that works with any SUT and CI/CD toolchain. It provides continuous automated testing based on a flexible architecture (users can author scenarios, jobs, and pipelines).

MSaaS is implemented with cloud-hosted microservices aligned with the NATO MSaaS Reference Architecture. A Scenario Editor is used to author relevant scenarios and a Scenario Repository stores and catalogs versioned scenarios. SADIE wraps existing Simulation Engines (STK, AFSIM, NGTS, OneSAF) into Simulation Service microservices.

The Automation & Orchestration ecosystem centers on the SBT Job Manager which coordinates execution of SBT jobs. The Job Manager connects Simulation Service Adapters (for configuration and control of Simulation Services) to SUT Adapters (to translate data into the required formats/schema and protocols for injection). It can be controlled from any build orchestrator (e.g., Jenkins) using a REST API.

We conclude with an end-to-end use case showing how SBT can use M&S to provide CI/CD quality gates based on regression testing results.

ABOUT THE AUTHORS

Jeremy Loomis is the Vice President of Engineering at NextGen, providing overall technical direction, guidance and mentoring across internal and external projects. Mr. Loomis has ~25 years of experience applying advanced technologies to meet the needs of diverse customers in commercial, civilian, DoD, and Intelligence Community (IC) organizations. Areas of interest include Modeling & Simulation, 3D Visualization, Artificial Intelligence, Software Factories, and Cloud Computing. He works with agile research teams to leverage COTS, GOTS, and FOSS technologies to develop innovative solutions using novel architectures and pragmatic software methodologies.

Alex Matthews is a Technical Director at NextGen, focusing on Digital Transformation related solutions. He evangelizes Digital Trinity concepts, including Digital Engineering, DevSecOps, Agile Transformation, Architecture Modernization, and Cloud/Kubernetes implementations. As a technical lead and architect, he is responsible for the technical execution of solutions across the systems/software lifecycles for AFLCMC, AFRL, DIA/NASIC, NAVAIR/NAWC, and Army PEO IEW&S. He has employed multiple M&S tools and technologies for these customers including AFSIM, STK, NGTS, EWIRDB, and TMAP.

Building a Cloud-Native Toolset for Flexible, Continuous, Automated Simulation-Based Testing

Jeremy Loomis, Alex Matthews
NextGen Federal Systems
Morgantown, WV
jloomis@nextgenfed.com, amatthews@nextgenfed.com

INTRODUCTION

The discipline of software testing is changing to align with the automated processes of Agile DevSecOps. In a DevSecOps environment, automated testing is executed by running test scripts or scenarios against the System Under Test (SUT) without human intervention. With many types of software (such as military applications), a challenging aspect of this testing is generating synthetic data and activity to stimulate the SUT.

Modeling and simulation (M&S) can help by providing representative test data from synthetic actors in a synthetic environment. However, current M&S tools and process operate in silos that are independent of the software systems lifecycle. As a results of this separation, M&S cannot be easily applied to the testing process to shorten acquisition cycles and accelerate deployment of software solutions to the field

To address this problem, we propose building on current M&S tools to implement a ***Simulation-Based Testing (SBT)*** approach which can inject synthetic test data into a Continuous Integration / Continuous Deployment (CI/CD) pipeline. Implementation of an SBT system requires two interconnected elements: M&S-as-a-Service (MSaaS) services and an Automation & Orchestration ecosystem.

BACKGROUND AND RELATED WORK

To provide context for our Simulation-Based Testing research, we first provide some background information on:

- Software Testing Activities
- Unique Testing Needs for Military Applications
- CI/CD and Automated Testing
- M&S-as-a-Service (MSaaS)

Software Testing Activities

Software testing takes many forms. Per (DoD CIO, 2021a), "Test activities may include, but are not limited to:

- unit test
- integration test
- regression test
- functional test
- system test
- performance test

All tests start with test planning and test development, which includes detailed test procedures, test scenarios, test scripts, and test data." As described in **Table 1**, these activities are implemented using ***test development*** tools, ***test data generation*** tools, and ***test execution*** tools.

Table 1. Testing Tools (DoD CIO, 2021a)

Tool	Features	Benefits	Inputs	Outputs
Test development tool	Assists test scenario, test script, and test data development. The specific tool varies, depending on the test activity and the application type.	Increase the automation and rate of testing	Test plan	test scenarios, test scripts, test data
Test data generator	Generates test data for the system (such as network traffic, web requests)	Increase test fidelity	Test scenario, test data	Input data for the SUT
Test tool suite	A set of test tools to perform unit test, interface test, system test, integration test, performance test and acceptance test of the software system.	Increase test automation, speed	Test scenario, test scripts, test data	Test results, test report

An SBT approach applies M&S technologies to support multiple test activities for all three of these tool categories:

- test development is done by assembling M&S scenarios, models, and environments
- test data is generated by running dynamic simulations with synthetic actors
- tests are executed by feeding the resulting data through a tool suite that can stimulate the SUT

CI/CD and Automated Testing

The discipline of software testing is changing to align with the automated processes of Agile DevSecOps. In a DevSecOps environment, automated testing is executed by running test scripts or scenarios against the System Under Test (SUT) without human intervention.

As shown in **Figure 1**, Test Tools (green outlines) are part of a CI/CD toolchain and are used to run automated tests on software that is built and scanned in a CI/CD pipeline. The test tool can be run in a "test environment" and/or "integration environment." To support rapid testing of multiple 'candidate builds', the deployment step for the SUT must itself be automated using Infrastructure as Code (IaC) techniques, such as automated provisioning and scripted installation, so that new SUT instances can be instantiated on-demand.

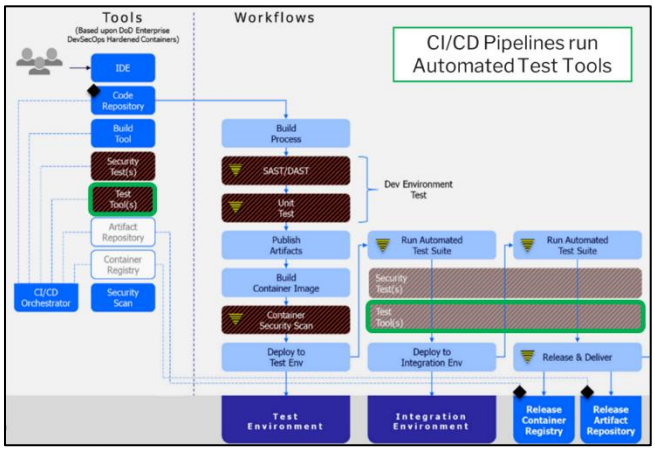

Figure 1. Automated Test Tools running in CI/CD Pipelines (adapted from DoD CIO, 2021b)

Unique Testing Needs for Military Applications

The Joint Staff established Joint Capability Areas (JCAs) as a standardized set of definitions that cover the complete range of military activities (Joint Staff, 2021). The JCAs are organized in a multi-level hierarchy covering 500+ distinct areas/domains. Some example JCA functions:

- JCA 2.3: "Battlespace Awareness | Processing / Exploitation"
- JCA 2.4: "Battlespace Awareness | Analysis, Prediction and Production"
- JCA 4.3: "Logistics | Maintain"
- JCA 5.3: "Command and Control | Planning"
- JCA 5.6: "Command and Control | Monitor"

DoD software systems support many of these military functions, in the form of intelligence processing tools, logistics information systems, command and control applications, etc. They are complex systems that process unique datasets and support a range of military tactics, techniques, and procedures (TTPs), such as Intelligence Prep of the Battlefield (IPB) or Air Battle Planning (ABP). This complexity requires extensive testing so that the software systems can be trusted for use by warfighters and intelligence analysts.

In addition to performing unique military functions, these software systems are often composed of multiple modules based on *Modular Open Systems Approach (MOSA)* principles. As such, testing can be needed at multiple architectural levels, to include System, Subsystem, Component, Service, and Application levels. The corresponding test activities and tools can also vary across these levels; for example, API tests can be an appropriate mechanism for verifying individual services that are composed into a larger system. Additionally, the data that flows through DoD systems often adheres to unique DoD schemas, formats, and protocols.

As an illustration, picture a ***notional all-source intelligence production system***. The system supports military function JCA 2.4 by providing intelligence analytics functions including data ingestion, content extraction, normalization, and correlation. In addition to user-facing applications, the system includes an "analytics service" that performs initial content extraction from available data. In this case, one testing goal would be to evaluate the analytics service for performance (speed and accuracy) as well as regression (consistency with prior versions). Data that is processed by the analytics service might include structured (e.g., MIDB), semi-structured (e.g., USMTF), and

unstructured (e.g., imagery/video) data. These data would represent the complex behaviors of friendly and adversary actors as reflected in intel feeds, sensor collection, blue force tracking, cyberspace actions, etc.

To properly perform software testing on such a service, 'representative' test data must be available that has:

- appropriate size/complexity (large data sets with realistic content)
- appropriate data formats (structured & unstructured text, motion imagery, still imagery)
- appropriate "signal-to-noise" ratio (to reflect non-perfect sources and networks)
- appropriate context/coherency (product of an operationally relevant scenario)

The testing process would involve feeding the test data into the analytics service, measuring execution time, and collecting the results. The results would be compared both to 'ground truth' (how did the service do at extracting expected entities from the synthetic data provided) and to 'prior results' (do extracted entities match those from a previous version of the analytics service).

M&S-as-a-Service (MSaaS)

According to the NATO MSaaS Reference Architecture (STO/NATO, 2019), MSaaS is defined as:

> "a new concept that includes service orientation and the provision of M&S applications via the as-a-service model of cloud computing to enable more composable simulation environments that can be deployed and executed on-demand. The MSaaS paradigm supports stand-alone use as well as integration of multiple simulated and real systems into a unified cloud-based simulation environment whenever the need arises."

Table 2 lists some of the MSaaS Architectural Building Blocks (ABBs) described in the Reference Architecture.

Table 2. Architectural Building Blocks (ABBs) for MSaaS. (STO/NATO, 2019).

M&S Enabling Services	
M&S Integration Services	Provide the infrastructure to connect producers and consumers of information and support an efficient and time-coherent exchange of simulation data
M&S Mediation Services	Provide broker and gateway services between incompatible producers and consumers of simulation-pertinent information
M&S Message-Oriented Middleware Services	Provide the capabilities for … exchange of data between producing and consuming Simulation Services, independent of data format and data content.
M&S Composition Services	Provide the capabilities to compose and execute a simulation from existing simulation services [using choreography or orchestration approaches].
Simulation Control Services	Provide the capability to provide input to a simulation execution, control the simulation execution, and collect output from the simulation execution.
Simulation Scenario Services	Provide the technical capabilities to define a scenario, to initialize the simulation environment with a scenario, and to handle simulation scenario events
M&S Information Services	Provides the capabilities to manage repositories of simulation service components and to manage references to [information required for execution]
M&S Repository Services	Provides the capabilities to store, retrieve and manage simulation resources and associations with / references to metadata managed by M&S Registry Services
M&S Registry Services	Provide the capabilities to store, manage, search and retrieve data about (i.e., metadata) simulation resources stored by the M&S Repository Services,
M&S Services	
Simulation Services	Set of capabilities for synthetic representation of (real-world) objects and events. Simulation Services are the service-oriented building blocks of simulations
Modelling Services	Category of services that encompass the entire suite of [tools], methodologies, … and publishing mechanisms needed to construct a Simulation Service
Composed Simulation Services	Results of composing simulation services. They offer entire simulations as services … that are generic enough to be used across many situations and/or occasions

Existing simulation packages offer modeling and simulation functions, but often require complex IT configuration to access the functions. By 'wrapping' those functions into **Modeling Services** and **Simulation Services**, the benefits

of a services-oriented architecture (SOA) can be realized. By adding an **M&S Registry**, **Simulation Control Services**, and **M&S Mediation Services**, several disparate capabilities can be orchestrated into cohesive workflows.

SIMULATION-BASED TESTING (SBT) APPROACH

The goal of SBT is to generate synthetic data and activity to stimulate the SUT. To be valuable for testing, the data must be representative of the data that is expected to be providing to the SUT in operational use. Depending on the goals for a given testing activity, the data required can be of varying fidelity, ranging from coarse inputs for basic interface testing, to more robust and high-fidelity datasets approaching a full "digital twin" (Roper, 2021) of the stimulating system for the SUT.

An SBT approach builds on current M&S tools, operating in a ***constructive-simulation*** mode. SBT leverages the proven ability of M&S to provide high-fidelity synthetic environments, synthetic systems (platforms, sensors), and synthetic actors (with complex sense-and-react behaviors) that can interact together without user intervention.

Using synthetic data (as opposed to data collected from the field or from controlled exercises), also allows testing of a system with 'edge cases': unique scenarios that could be difficult to capture in real life, but that are of keen interest to military planners. Simulating dynamic and unique use cases can give developers additional confidence that the operational system will robustly handle a range of scenarios when fielded.

Example M&S tools that could be used by an SBT system include GOTS simulation engines such as OneSAF and AFSIM that allow for the modeling of multi-domain sensors (space and high altitude, aerial, and terrestrial layer) as they attempt to detect and identify multiple dynamic targets. The simulated targets have representative attributes and relationships and exhibit realistic behaviors through the use of Computer Generated Forces (CGF) algorithms. COTS simulation engines such as STK could provide additional physics-based modeling of geodynamics and sensor performance (for example, detailed geometric and radiometric models of aerial and space ISR systems).

With SBT, the representative test data (from synthetic actors in a synthetic environment) is provided to the SUT using existing interfaces and protocols. In cases where the SUT operates on 'real time' information, the SBT system must be able to deliver the test data with realistic latency, velocity, and timing.

The goal for SBT is to integrate into the CI/CD process for the SUT software as another 'automated test tool' alongside existing tools such as UI test tools (e.g., Selenium), API test tools (e.g., Soap UI), and performance tools (e.g., LoadRunner). An SBT approach can be used in isolation or in concert with these other tools. For example, synthetic data can be fed into the SUT while a UI test is performed to simulate operator actions.

Desired system attributes for a SBT system:

- ***Cloud-Native***: Scalable deployment to any Kubernetes cluster; supports public/private clouds
- ***Modular***: Modular system works with any SUT and any CI/CD Factory with varied tech stacks
- ***Flexible***: Allow users to author their own SBT scenarios, jobs, and pipelines
- ***Continuous***: Tied into the CI/CD process for 'continuous testing' enabling rapid deployments and fielding
- ***Automated***: Simulation execution does not require 'man-in-the-loop' steps during testing
- ***Cross-Functional***: Brings M&S engineers and SMEs directly into the software lifecycle

PROOF-OF-CONCEPT SBT TOOLSET

We now describe our ***Simulation And DevOps Integration Environment (SADIE)*** project, which built a proof of concept SBT toolset using existing COTS/GOTS software. SADIE is a cloud-native (deployable to any Kubernetes cluster) modular toolset that works with any SUT and CI/CD toolchain. It provides continuous automated testing based on a flexible architecture (users can author scenarios, jobs, and pipelines).

Figure 2 illustrates the logical system architecture for SADIE, consisting of an Analysis Ecosystem, an Automation Ecosystem, and an MSaaS Ecosystem working together to provide SBT of a SUT Instance.

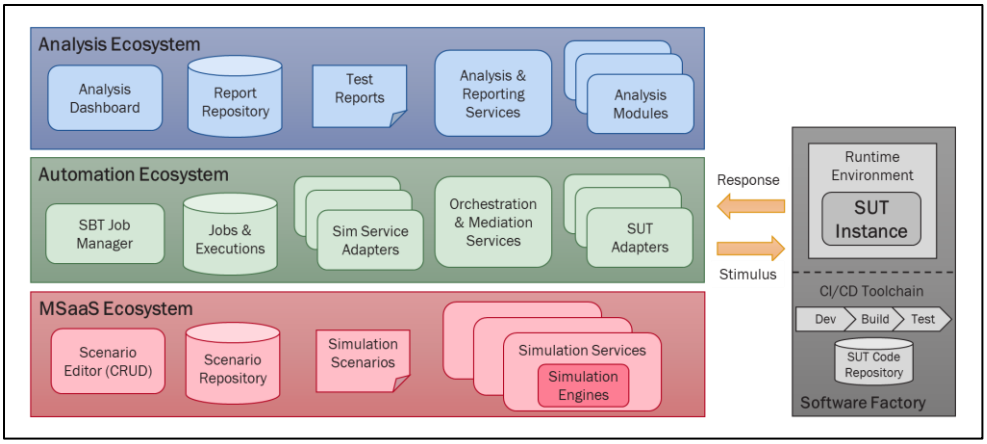

Figure 2. Logical System Architecture for SADIE

SADIE is implemented with cloud-hosted microservices aligned with the NATO MSaaS Reference Architecture. The system design for SADIE focuses on the following ABBs:

- M&S Mediation Services: included as *Sim Service Adapters* in the automation ecosystem as part of SBT jobs
- Simulation Control Services: each simulation service offers control endpoints used by the job manager.
- Simulation Scenario Services: the overall simulation scenario is defined by the SBT job
- M&S Repository/Registry Service: the *Scenario Repository* will hold M&S resources and metadata tags
- Simulation Services: a key element of SADIE is wrapping existing *Simulation Engines* into services
- Modelling Services: in the current version of SADIE, modeling occurs in desktop *Scenario Editor* tools

The *SBT Job Manager* is the central coordination point for the end-to-end SBT lifecycle. As an SBT job is executed, *Simulation Services* are launched, simulation inputs/outputs are configured, data is streamed to/from the SUT, results are analyzed, and analysis is performed in support of testing goals.

The SUT Instance is hosted in an appropriate runtime environment (potentially distinct from the runtime environment for SADIE itself). During execution of a SBT job, the SUT Instance is stimulated by the SADIE system to collect results. Depending on the type of test activity being executed, the SUT Instance itself can be 'ephemeral' (for example, when running SBT-based regression tests against a 'candidate' build produced by a developer branch in the SUT Code Repository).

MSaaS Ecosystem

For SADIE, a *Scenario Editor* is used to author relevant scenarios. We are using existing desktop scenario editing tools provided for each core simulation technology, such as AFSIM Wizard, OneSAF MTC and STK Desktop.

The *Scenario Repository* will store and catalog versioned scenarios. The Repository will be built on an open-source data management framwork. This will allow for a repository with CRUD (Create, Retrieve, Update, Delete) functions defined through REST API interfaces. For the initial version of SADIE, we store scenario resources in a simple web server to allow access from the SBT Job Manager.

SADIE wraps existing *Simulation Engines* (STK, AFSIM, NGTS, OneSAF) into *Simulation Service* microservices. As shown in **Figure 3**, a Simulation Service microservice provides a standardized interface that includes functions for simulation control, clock management, data input, and data outputs. The REST API for each Simulation Service is defined using the OpenAPI (formerly SwaggerUI) specification to allow for self-documenting and easy to implement service endpoints.

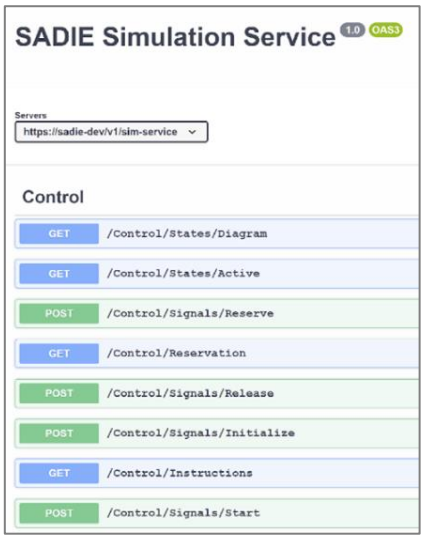

Figure 3. Each Simulation Service implements an OpenAPI interface

Automation Ecosystem

The Automation ecosystem centers on the *SBT Job Manager* which coordinates execution of SBT jobs. The Job Manager connects Simulation Service Adapters (for configuration and control of Simulation Services) to SUT Adapters (to translate data into the required formats/schema and protocols for injection).

Orchestration & Mediation Services are used within job steps to achieve stimulation (injecting data into the SUT) and data collection (retrieving data from the SUT). Each SBT Job includes a test stimulation driver and a mechanism to record test outcomes. The SADIE Job Manager also includes several example 'SBT job templates' that can be customized for new use cases.

Data injection into a SUT typically follows one of three pathways:

- delivery of data via 'sensor feeds' (e.g., GEOINT, MTI, SAR, SIGINT, FMV, IBS, USMTF)
- providing data using 'interoperability interfaces' (e.g., JMPS, CoT, PASS, ISA, DDS)
- inserting data using a 'system training interface' (as used by IEWPTPT, for example)

Collecting data from a SUT (raw and processed) follows similar patterns:

- gathering 'export files' created by the SUT (such as Excel, free text, or USMTF documents)
- connecting using 'interoperability interfaces' (e.g., MIDB or OGC data discovery services)
- direct access to 'internal databases or APIs' (such as the Army's Tactical Entity Database)

Analysis Ecosystem

Within SADIE, *Analysis & Reporting Services* create test reports based on the results from an SBT job. The Analysis Modules compute metrics such as accuracy, performance, and capacity. For example, to compute system accuracy for feature extraction functions of a SUT, an analysis module can compare extracted features (produced by the SUT in response to synthetic inputs) to a ground-truth "answer key" represented by the simulation.

SADIE is also designed to allow plug-ins that provide 3rd party analytics for different forms of V&V. To be integrated into SADIE, an analysis plug-in is typically provided as a container image that can be instantiated by the Job Manager. At runtime the container is connected to data outputs generated by the Simulation Service and (optionally) to 'reference' data configured within the SBT job. The analysis is triggered by the Job Manager and the results are collected as artifacts from each SBT job execution.

The resulting reports and metrics can then be collected in a database and presented in an intuitive dashboard. For example, existing CI/CD dashboards such as Jenkins Job dashboards and GitLab CI Pages can be used to associate SBT analysis results with a specific pipeline execution.

CI/CD Integration

To allow SADIE to be integrated within a CI/CD toolchain, the SBT Job Manager and Analysis Services can be controlled from any build orchestrator (e.g., Jenkins, as in **Figure 4**) using a REST API.

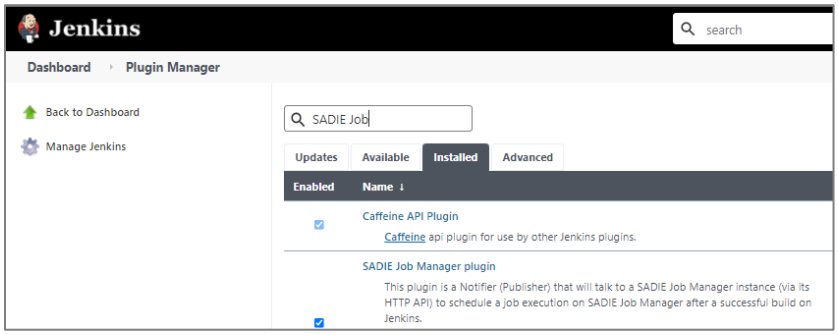

Figure 4. Installing the SADIE plugin into a Jenkins CI/CD Server

This integration allows for a CI/CD pipeline to now include customized SBT steps/stages that run as part of an overall pipeline execution process. For example, after the CI/CD pipeline compiles, scans, and deploys the SUT to a test server, the SBT job can be launched to execute runtime SBT testing against the SUT instance.

Cloud Hosting

SADIE can be deployed into any Cloud Native Computing Foundation (CNCF)-compliant Kubernetes platform (such as AWS EKS or Rancher RKE2). SADIE's components and microservices (defined via OCI container images) are run in multiple Kubernetes pods, which can be distributed across a multi-node Kubernetes cluster. This requires that any Simulation Engines being used by SADIE be deployable into base containers before service-wrapping.

With this Kubernetes-based architecture, SADIE is portable and scalable for various environments: the cluster can be instantiated using fixed (on-prem) servers or in a commercial cloud environment (such as AWS).

CASE STUDY

We conclude with an end-to-end use case showing how SBT can use M&S to provide CI/CD quality gates based on regression testing results. **Figure 5** provides an overview of the SBT process for the case study.

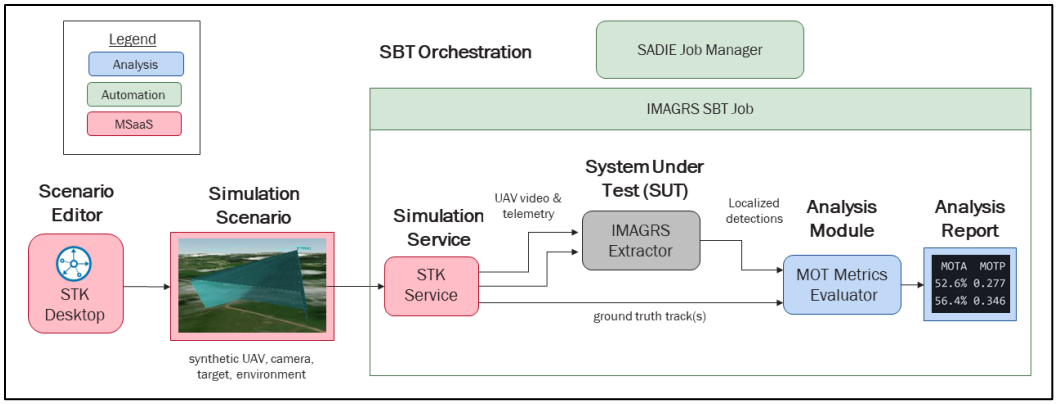

Figure 5. Overview of SBT Case Study

The various components of the case study are further described in the sections below.

System Under Test (SUT)

The *Intelligent Multirotor Autonomous Ground Relocatable Sensor (IMAGRS)* system developed by NextGen Federal Systems is an advanced small Unmanned Aerial System (sUAS) for persistent autonomous surveillance. It has an integrated EO/IR payload and includes state-of-the-art onboard Machine Learning (ML) and Computer Vision (CV) algorithms running on a small-board computer (SBC).

The "IMAGRS Extractor" is the subsystem of IMAGRS that processes a georeferenced video stream and attempts to detect and localize targets of interest in the video scene.

- Inputs: timestamped UAV video frames and platform telemetry (GPS location, IMU orientation)
- Output: timestamped target detections (time, latitude, longitude, altitude)

The Extractor subsystem is implemented as a Python library (wrapped in a command-line program for testing).

Using SBT to test the Extractor module in isolation using synthetic video data reduces the need to perform full integration testing with the IMAGRS hardware platform. To simplify subsystem testing, the Extractor was wrapped in a simple Docker container so that the SUT could be easily deployed by an SBT job for stimulation (providing required inputs) and collection of results (capturing target detections).

Desired Testing

For this case study, the primary desired testing was functional testing to measure 'accuracy': to determine how well the Extractor can correctly detect and locate targets in a synthetic video stream. An alternate metric would be to measure performance (how long does the system take to perform localization), since latency in this stage influences the downstream performance of other IMAGRS components for target tracking and mission planning.

Either of these tests could be used for determining if a regression has occurred: 'does build N of the system provide the same/better accuracy and performance compared to build N-1?' For this case study, which focused on functional accuracy, a regression was judged to occur when build N detected fewer targets than the previous build.

Scenario Editor

The Systems Tool Kit (STK) (AGI, 2022) is a digital mission engineering application that "features an accurate, physics-based modeling environment to analyze platforms and payloads in a realistic mission context." STK was selected because of its ability to analyze complex systems with a focus on their operational environments.

For this case study, we needed a realistic and time-dynamic three-dimensional simulation that included high-resolution terrain, imagery, and precise models of ground and air platforms. With STK, we could simulate the entire system-of-systems in action, at any location and at any time, to gain a clear understanding of its behavior and mission performance.

Simulation Scenario

A synthetic environment was created in STK using terrain data and satellite imagery. In the environment, two moving entities were simulated:

- A synthetic model of the UAV with a representative field of view (FOV)
- A synthetic model of a target vehicle (truck) driving on the terrain surface.

Using the STK 3D graphics capability, one window was designated as the virtual 'camera view': the 3D camera was tethered to the boresight of the virtual camera and during scenario animation, this window was used to collect a series of synthetic video frames. At the same time, an STK report was used to capture telemetry for both the UAV system (including position and camera orientation), and the target vehicle (to be used as 'ground truth').

An STK visualization of the overall scenario, including the virtual camera, is shown in **Figure 6**.

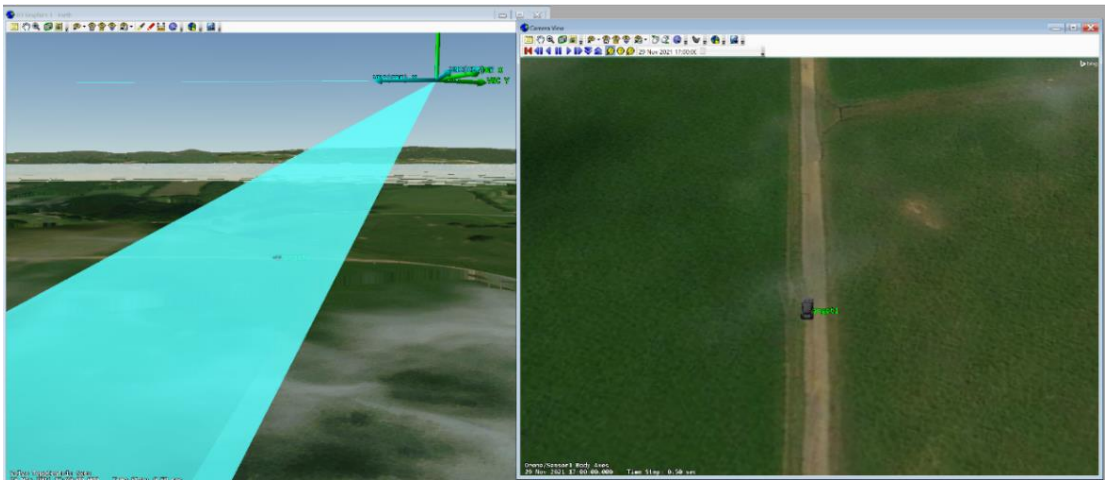

Figure 6. Visualization of the simulation scenario used for Extractor testing

Simulation Service

The STK Engine Java library was wrapped into a RESTful *STK Service* running in Apache Tomcat. The STK Service implemented the required SADIE endpoints (for simulation control, clock management, data input, and data outputs). The web service was then containerized into a Docker image by adding the service WAR to a Tomcat base image.

The REST invocation sequence used by the SBT job to execute the simulation was:

1) POST Control/Signals/Reserve
2) GET Control/States/Active [poll until State = 'Reserved']
3) POST Control/Signals/Initialize;
4) GET Control/States/Active [poll until State = 'Active:Ready']
5) POST Control/Signals/Start
6) GET Control/States/Active [poll until State = 'Idle']

This sequence of calls did the following: initialized the STK service with the selected scenario file, specified which outputs were desired, began the simulation/animation, and waited until the simulation was complete.

Analysis Module

For this case study, we used the *py-motmetrics* library (Heindl, 2022), which provides a Python implementation of metrics for benchmarking multiple object trackers (MOT). It measures "the performance of multiple object trackers [using] CLEAR-MOT metrics and ID metrics. Both metrics attempt to find a minimum cost assignment between ground truth objects and predictions." This provides an appropriate metric for comparison of the synthetic target vehicle track and the localized detections produced by the IMAGRS Extractor.

Since we were interested in just detection (not identification), the detection/tracking metrics listed in **Table 3** applied.

Table 3. Metrics for object detection and tracking

Name	Description
num_frames	Total number of frames.
num_matches	Total number matches.
num_switches	Total number of track switches.
num_false_positives	Total number of false positives (false-alarms).
num_misses	Total number of misses.
num_detections	Total number of detected objects including matches and switches.
num_objects	Total number of unique object appearances over all frames.
num_predictions	Total number of unique prediction appearances over all frames.
num_unique_objects	Total number of unique object ids encountered.
mostly_tracked	Number of objects tracked for at least 80 percent of lifespan.
partially_tracked	Number of objects tracked between 20 and 80 percent of lifespan.
mostly_lost	Number of objects tracked less than 20 percent of lifespan.
num_fragmentations	Total number of switches from tracked to not tracked.
motp	Multiple object tracker precision.
mota	Multiple object tracker accuracy.
precision	Number of detected objects over sum of detected and false positives.
recall	Number of detections over number of objects.

SBT Job Structure

The high-level workflow for the SBT job in this use case is shown in **Figure 7**. In Step #3, the Sim Service (STK Service) is initialized with the pre-prepared Scenario along with 'instructions' to publish UAV video, telemetry, and ground tracks to the Workspace. This allows for the data exchange with the SUT (IMAGRS Extractor) in Step #7 and with the Analysis Module (MOT Metrics) in Step #10.

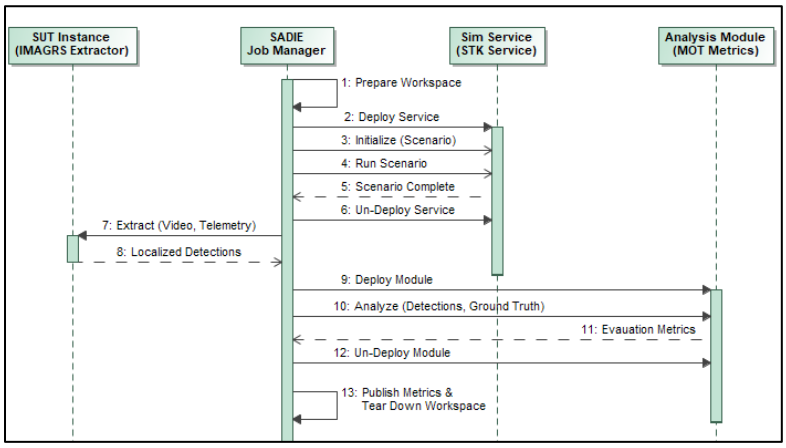

Figure 7. High-level sequence diagram for IMAGRS SBT Job

Build Orchestration

The entire end-to-end flow for SBT testing of the IMAGRS Extractor was orchestrated by a Jenkins pipeline (**Figure 8**) that deployed the SUT, ran the SBT job (including analysis), and collected the analysis results.

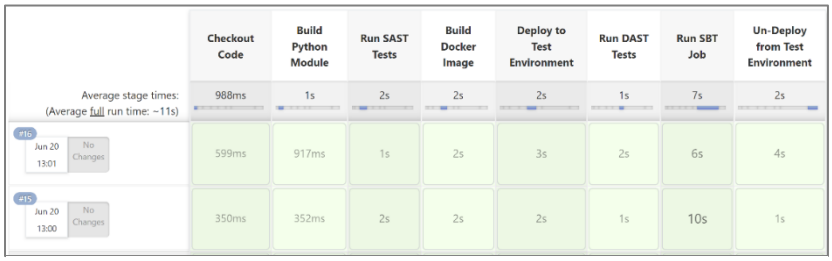

Figure 8. Notional Jenkins CI/CD pipeline with an SBT stage

The overall CI/CD process is initiated when a developer checks in code that modifies part of the SUT (IMAGRS Extractor). At this point, Jenkins detects the code check-in and initiates the CI/CD pipeline. The pipeline includes stages that build the code, run static (SAST) code checks, build a Docker image, deploy the container to a test environment, and run dynamic (DAST) tests. During the "Run SBT Job" stage, the following occurs:

1. The SUT Instance is exposed to the SBT Job Manager
2. The SBT Job is launched, and the CI/CD pipeline waits for completion
3. The SBT Job results (MOT Metrics) are published as CI/CD artifacts

When the CI/CD pipeline completes, the MOT Metrics are available in the Jenkins dashboard as a custom HTML page. Optionally, a check can be added to the pipeline that compares the MOT Metrics to prior results to detect any potential differences (if the current build does not result in the same detection outputs). In most cases, the change would be considered a 'regression error.' Based on this, the metric check can be used as a "quality gate:" if a regression occurs, the pipeline can be "failed", alerting the developer to investigate the root cause of the error.

By integrating SBT testing into the overall build orchestration process defined in a Jenkins pipeline, the SBT capability adds an additional dimension to other automated tests that are run as part of the CI/CD pipeline. All test categories are triggered by each code change that is committed during SUT development.

CONCLUSION

Results

For IMAGRS, the SADIE SBT capability provided a unique ability to evaluate the Extractor subsystem without integrating the software into the onboard computer and taking the sUAS into the field for testing. This saved numerous staff hours of travel to the testing facility and reduced the level of physical risk to the platform by reducing the number of flight hours required to evaluate the target detection functions of IMAGRS,

Based on this case study and other similar use cases, SBT shows great value as a new technique for automated testing of military applications. The SADIE proof-of-concept has shown the feasibility of using an SBT system to apply powerful M&S technologies in a new way.

Some of the anticipated benefits from SBT that were demonstrated by this project include:

- Making subsystem testing feasible without full system deployment to the field
- Providing quantitative and repeatable metrics of subsystem performance
- Ability to perform high volumes of testing using different variations of synthetic environments/actors.

Future Work

Some areas for future capabilities of the SADIE proof-of-concept system include:

- Implementation of the Scenario Repository component
- Documented SDK for implementing new Scenario Services
- Genericizing each step of the SBT workflow to allow for easier authoring of SBT jobs

We are ready to work with stakeholders in the M&S and software testing communities to show how SBT can add value to automated software testing.

REFERENCES

AGI (2022). *AGI: Systems Toolkit (STK)*. Retrieved from https://www.agi.com/products/stk

DoD CIO (2021a). *DevSecOps Fundamentals Guidebook: DevSecOps Tools & Activities*. September 2021, Version 2.1. Retrieved April 22, 2022 from https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps Fundamentals Guidebook-DevSecOps Tools and Activities_DoD-CIO_20211019.pdf

DoD CIO (2021b). *DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes*. March 2021, Version 2.0

Heindl, Christoph (2022). *Py-motmetrics Readme*. Retrieved from https://github.com/cheind/py-motmetrics.

Joint Staff (2021). *Charter Of The Joint Requirements Oversight Council And Implementation Of The Joint Capabilities Integration And Development System*. CJCSI 5123.01I, 30 October 2021

Roper, Will (2021). *Bending the spoon: Guidebook for digital engineering and e-series*. United States Air Force. https://www.af.mil/Portals/1/documents/2021SAF/01_Jan/Bending_the_Spoon.pdf

STO/NATO (2019). *Modelling and Simulation as a Service, Volume 1: MSaaS Technical Reference Architecture*. TR-MSG-136-Part-IV. May 2019