Introduction

Creating a chatbot proof of concept is easy. If you're feeling inspired, you can pretty quickly build something that can hold simple conversations. There are a lot of tutorials that walk you through the basics of building AI assistants.Going to production, on the other hand, is not so simple. Many enterprises experience challenges in production-ready systems that have large user bases, several third-party applications, security and privacy mandates, and polyglot environments.

This whitepaper walks through the challenges of taking a chatbot proof of concept to production. We'll explain the process of building robust chatbot applications, list some of the challenges uncovered when going to production, and discuss successful techniques to scale chatbot applications.

Building a Minimum Viable Assistant (MVA)

A minimum viable assistant is something that fulfills its core purpose and basic functionality. If you're building a help desk assistant, an MVA version is something that can answer some basic questions or execute the most frequently requested task or two. It may not answer every question or do a lot of things but, in essence, it has to be well-defined and tested.

The build process can be a little overwhelming at first, so start with a few simple questions:

First, what do you want your assistant to say and do, i.e., what is its purpose? It's important for your assistant to answer questions but it's more exciting and useful if it also does things for your users. If you're building a help desk assistant, your goal might be to automate 30% of conversations. That means you want your assistant to answer frequently asked questions and resolve routine technical issues; for example, if 20% of tickets are for password resets, that might be an important task to automate. If you're building a customer service assistant for a retailer, your goal might be to decrease cart abandonment rate. That means you want your assistant to answer questions about products. You might also want to make a list of all the back-end systems your assistant might interact with and start to plan your integrations. Start small, preferably with ten

questions or so and one or two tasks the assistant can execute on the user's behalf, and grow from there.

Second, what do you want your assistant to sound like, i.e., what is its tone and personality? In the case of either the help desk or retail assistant, the assistant's goal is to be helpful; its tone and personality need to reflect that.

Most conversational AI applications have a similar workflow, so it helps to think about:

- What you want the end-user to accomplish
- How you want your assistant to sound, and design dialogue accordingly
- What system actions you want to execute

Finally, one of the best things you can do to build a robust conversational AI application is to treat it like any other software application and follow best practices. In the case of either the help desk or retail assistant, you'll want to follow SOLID principles, secure coding practices, implement CI/CD, and ensure decent code coverage.

The next step is to release the initial MVA first to internal test users and then to external test users. User testing provides a lot of valuable feedback and insight into user behavior that you can use to guide design modifications, enhancements, and other changes to the product. The process of listening to user insights and using them to improve your assistant is known as conversation-driven development (CDD) and it is the best way to build something that scales reliably. In addition, user testing helps you build a representative data set which can be annotated and turned into training data. This is important because hypothetical conversations don't model the way real users interact with your assistant and should therefore not make up more than 10% of your data set.

It's also important to ensure that test users don't have prior knowledge of the assistant because they tend to stick to the conversation flows they know your assistant can handle; a user without any knowledge of the assistant will interact with it in ways you can't anticipate, thereby preparing it for production.

All of the aforementioned steps we've talked about up until this point get you to a minimum viable assistant. The next step is to release it to real users aka production. Deploying your assistant to production makes it available to real users. It's important to do this incrementally; i.e., release it to 10% of your audience, then 20% and so forth. It allows you to test your assistant's validity, collect real conversations to improve your

assistant, and measure its performance among other things, while still giving you the opportunity to course-correct should you find that the initial audience is not receptive to your assistant. However, deploying to production isn't always easy. So let's talk about some of the common challenges we face in deploying and maintaining conversational AI applications in production.

Challenge 1: Security Mandates

For a conversational AI application to be in production in an enterprise setting, it's imperative that it meet enterprise security standards. Security mandates can be particularly challenging as they generally deal with all the layers of the Open Systems Interconnection (OSI) model, i.e., the entire hardware and software stack.

Rasa Open Source and Rasa Enterprise, a platform that delivers scalable user-centered assistants, are built on a microservices architecture. This means that there are several loosely coupled services that make up the assistant's architecture. Each of these services require specific configuration settings and credentials to speak with other services. As with any other software application, as the number of configurations and credentials that you have to manage increase, the complexity and difficulty of deploying the application increases as well.

Enterprise security mandates dictate how Rasa's configurations and credentials should be securely stored and retrieved. These mandates specify the file permissions that need to be set on the configuration files. They also tell you how to encrypt and decrypt credential details.

The methodologies used to adhere to these security standards depend on the type of infrastructure or platform used to deploy the assistant. Some platforms like Kubernetes come with out-of-the-box support for injecting credentials and environment variables in a secure manner, while other platforms like Bare Metal and VM might need custom solutions. Some enterprises might mandate a service like Vault to manage sensitive configurations and credentials. Using Vault in itself will add an additional layer of complexity in deploying the assistant.

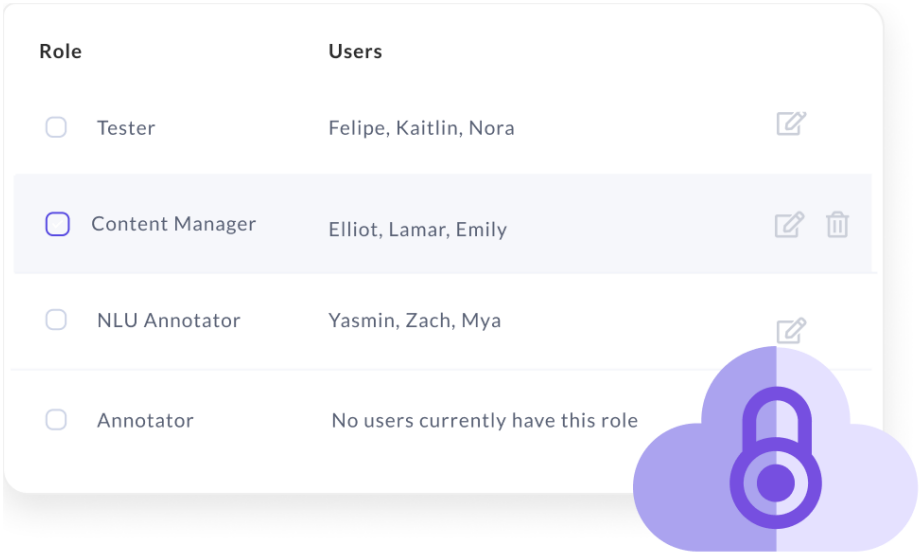
In addition, the security mandates specify how to secure data in motion. Securing data in motion is typically done using transport layer security (TLS) and HTTPS when services communicate with each other and with the outside world. If TLS is a mandatory

requirement, certificate management will become yet another thing that you'll need to configure and manage. If Kubernetes is the platform of choice to deploy the assistant or if you're using cloud providers like Openshift, AWS, Google Cloud and so forth, you might want to use a package like cert-manger to ease certificate management. Another common strategy used in Kubernetes environments is to expose Rasa services via ingress, that has OOTB support for TLS termination and service load balancing.

Enterprises may also require that the conversation data be secured at rest. Since data persistence in Rasa is typically delegated to databases, the built-in encryption capabilities of the databases should be employed to secure the conversation transcript data at rest.

When it comes to authentication and authorization, Single Sign-On (SSO), often accomplished using Lightweight Directory Access Protocol (LDAP) or Security Assertion Markup Language (SAML), is a de facto standard in most enterprises. In the case of SAML, enterprises typically deploy an identity provider. Rasa Enterprise, a version of Rasa X for enterprises building and deploying AI assistants at scale, supports SAML based SSO. Enabling SSO requires securely providing and managing X.509 certificates, private keys, and configuring the identity provider service URL, bindings, and other connection details. You may want to use enterprise security mandates and standards as guiding principles to handle and manage your SSO configurations.

If you're using Rasa Enterprise, it's important to know that Rasa adheres to the principles of least privilege in the form of role based access controls. Manage your roles and permissions from the "Manage Roles" page to ensure that enterprise users are given appropriate access privileges.



Challenge 2: Integration with Third-party Services

It can be pretty exciting to see your assistant answer questions. But getting your assistant to execute actions on your user's behalf is one of the most rewarding things when you're designing and building conversational AI applications. Organizing third-party service integrations based on your assistant's use-case and domain can be cumbersome and overwhelming at times.

With Rasa, you can leverage this capability via custom actions. A custom action is a hook that allows you to execute arbitrary code in response to a user message. Custom actions can turn on the lights in your home, add an event to a calendar, check a user's bank balance, or anything else you can imagine. Custom actions can be used to integrate your assistant with third-party services via REST API calls.

Establishing connections to these third-party services depend on a few things:

- Enterprise mandates
- Deployment topology
- Firewall and network rules

You may have to use HTTPS proxy, HTTP authentication header with JSON Web Token (JWT), and additional HTTP request signing with certificates, while interacting with third-party services.

As always, be sure to not hardcode or store third-party service configuration and connection details as part of your source code; securely store these values in external configuration files that are set up during deployment. You may use purpose-built secret data management services like Vault to store and retrieve third-party service configuration and connection details.

In addition, Rasa recommends that you follow the single responsibility principle (SRP) when you create custom actions. This means, it's best to not perform too many things under a single custom action. For example, if you want your help desk assistant to open, follow up on, and close incident management tickets, a single custom action should not execute all of these tasks; ideally, you'll create a separate custom action for each of these tasks.

To partially automate the process of creating custom actions, use the Rasa Software Development Kit (SDK). Rasa provides a Python SDK to create custom actions. You also have the flexibility to write this in any other language. The SDK does a few useful things:

- It reduces the boilerplate code required to set up a REST API-based actions service; this service can be scaled up or down independent of the Rasa service
- It allows you to easily modify the tracker and set slot values based on third-party service call results
- It can be used to send one or more responses back to the user

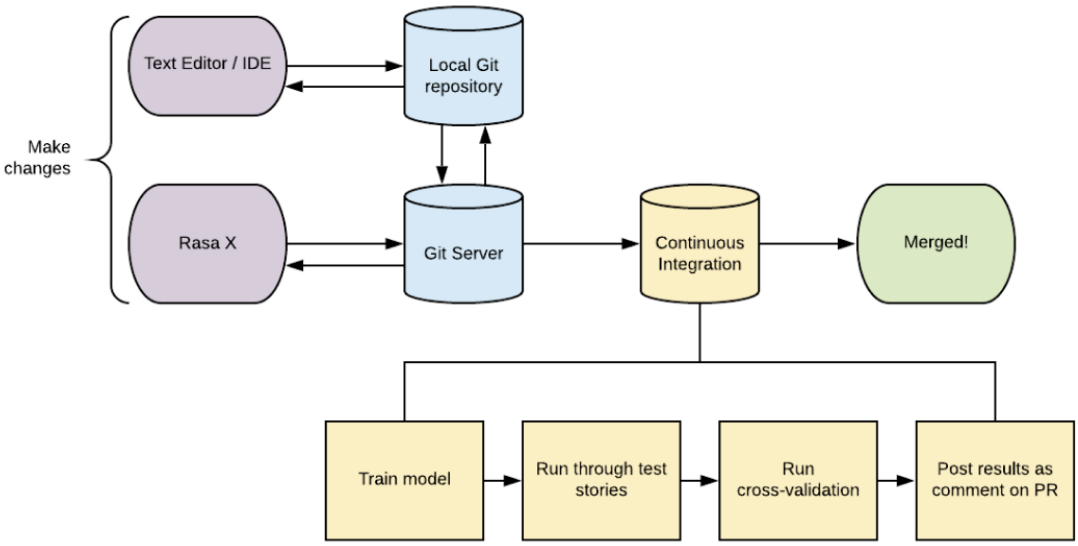
Before implementing custom actions, plan the integration points based on your assistant's use case and domain. In addition, study user behavior and user-engagement signals, like what users are looking at or clicking on to learn about their needs. For example, if 20% of users ask if your assistant can make a credit card payment, then it may make sense to implement this capability. A common pattern is building these skills as question-answers first; for example, provide users with a deep link on how they can make a credit card payment, and then based on traction, implement said capability as an actual user goal. Optimizing your integration strategy based on what users actually need and use is a great way to build an assistant that can really help users.

Challenge 3: Varying DevOps Expertise

DevOps is a set of practices, culture, and tools that combines software development and IT operations to deliver and maintain software applications in production. It streamlines the process building, testing, and deploying software applications. DevSecOps, a variation of DevOps, is when you include security as a critical part of your conversational AI life cycle. This typically involves auditing security guidelines, running vulnerability scans, and proactively thinking about your infrastructure and data security.

As you can probably guess, DevOps is pretty important when it comes to delivering production-grade AI assistants. However, there's a wide variation in level of DevOps expertise and lack of DevOps tools that can cause long delays in getting an assistant to production.

Rasa Enterprise enables you to run automated tests, merge code updates, train models, enable multi-disciplinary team collaboration, and automate similar DevOps processes to build scalable conversational experiences. Rasa also recommends that you plan ahead and involve the various DevOps stakeholders early in the process to ensure a smooth deployment to production.



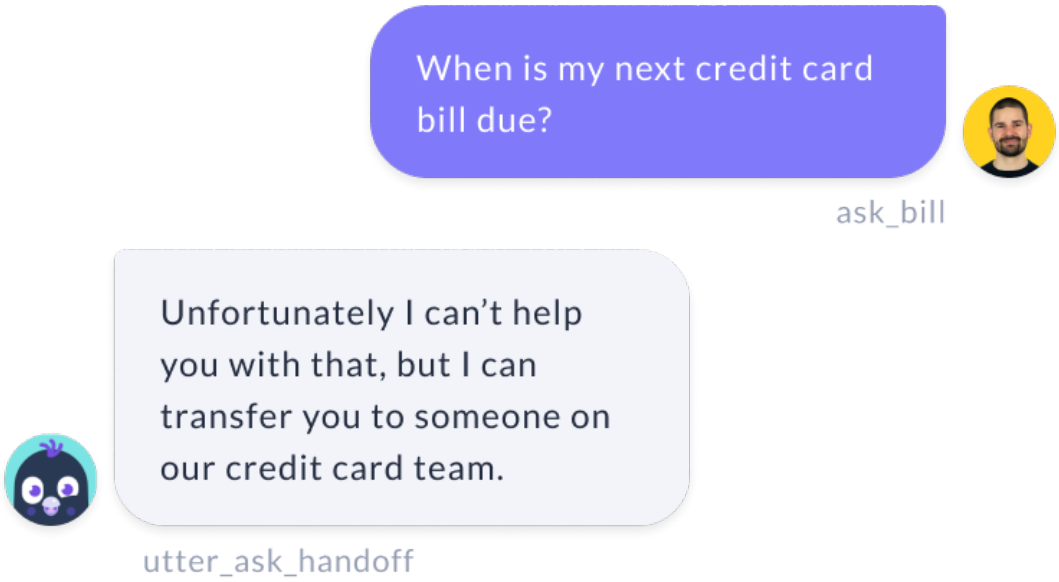
Challenge 4: Fallback

It's important to ensure that your assistant answers the questions it's supposed to, i.e., your assistant should correctly identify all of the defined intents. It's also equally important that your assistant respond appropriately to questions it doesn't know the answer to, i.e., it should fail gracefully. In production environments, your assistant might not respond to all questions, especially in the beginning. When this happens, stakeholders that are unfamiliar with conversational AI processes might get concerned.

However, this is a common occurrence and there are a few ways to solve this problem. First, ensure that your assistant provides a fallback response; if the user provides an ambiguous response, try to disambiguate it by prompting the user to say it a different way or let the user know the assistant can't answer the question as it's still learning.

Second, fix unhandled intents or unanswered questions by adding it to the assistant's knowledge base if it's within scope. Third, ensure that the assistant returns specific fallback responses to questions. For example, if the assistant can't answer questions about credit card bill payments, instead of returning a generic fallback response, an appropriate fallback response might be asking to transfer the user to a human agent or

someone on the credit card team. Monitor your conversations on a daily or weekly basis and make the appropriate fixes to make fallback more specific.



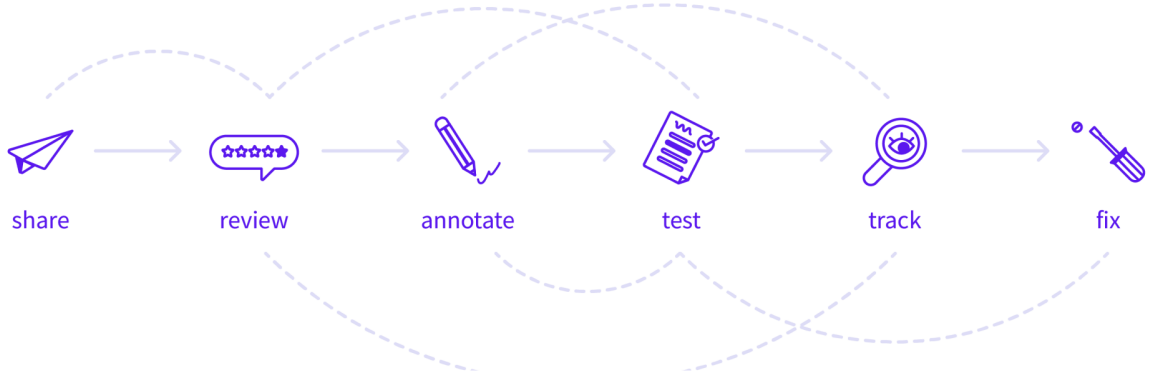
Challenge 5: Scaling Conversations

One of the challenges with building resilient AI assistants is that it's impossible to anticipate all of the things users could say to it. This is especially true when the assistant's scope or user base expands.

As more users start interacting with your assistant or your team starts to support additional capabilities and conversation paths, the chances that your assistant mishandles user requests become understandably higher. You still need to ensure your assistant adapts to user requests and behavior, as opposed to forcing the user to adapt to pre-programmed conversation paths. In addition, you'll need to train your model on what users actually say, instead of only relying on synthetic datasets. Synthetic or developer-generated datasets don't often match what users actually say in production; therefore, models trained on user-generated data are more accurate and performant. This can be hard to scale.

However, following a user-centric approach like CDD where you listen and adapt to user needs can mitigate some of these challenges. One of the things that sets conversational AI apart from other software applications is that when talking to chatbots, users are telling you exactly what they want. They're also communicating their intention in ways you haven't thought of or included in your dataset. This is why it's critical to collect, review, annotate these insights and iteratively improve your assistant.

Rasa Enterprise allows you to do CDD. It's important to keep in mind that this isn't a linear process and you'll have to jump between each of the actions defined within CDD and make continuous improvements to your assistant. Over time, this process ensures that your assistant is adapting to user needs, and actually getting better over time.



Conclusion

Building resilient AI assistants is not always easy. Putting together a proof of concept is simple enough but scaling at the enterprise-level is a different story. Some of the challenges enterprises experience are:

- Security mandates
- Integration with third-party systems
- Proper DevOps tools
- Scalability as it relates to conversations

In this paper, we discussed some effective strategies to overcome these challenges and walked through successful techniques to scale conversational AI applications. With considerate design, and collaboration between your users and development teams, you can build AI assistants that can provide engaging experiences and truly help users.