

Day 4: Deploying and improving your assistant using Rasa X

Agenda

Day 1: Deep dive into NLU and dialogue management with Rasa Open Source

Day 2: Create an MVP assistant, end-to-end testing, DIET architecture, and TED policy

Day 3: Adding custom actions and implementing forms

Day 4: Deploying and improving your assistant using Rasa X

Day 5: Recap + Q&A Panel

How to get help

- Please ask your questions in the **#workshop-help** Slack channel rather than the Zoom chat. Slack is the place the Rasa team will be monitoring most closely.
 - Karen, Mady, and Juste will be in Slack answering questions, as well as Arjaan, Ella, and Greg from our Customer Success Engineering team
- Monday - Thursday the Rasa team will be dedicating time to answering your questions in Slack from 4pm - 6:00 pm Central European Time
 - Feel free to ask questions outside of these hours, but responses may be a little slower
- A note on time zones:
 - The Rasa team is based across the US and in Berlin. We'll do our best to answer questions within team members' working hours, but please keep in mind, some discussions may need to take place async rather than in real time.

What we've set up for today...

The Rasa team has provided a virtual machine on Google Cloud Platform and installed Rasa X.

- Check your Slack DMs for Rasa X login details and the URL to access the server
- Go ahead and make sure you can log in and access the server via SSH

You have built a minimum viable assistant this week

- Thanks Mady & Juste!

The rest of the other things we will need are in place

- financial-demo repo on GitHub - forked to your personal account
- Telegram account ready to go

Today's theme: deployment and production

- Lecture/Instructor Demo: setting up the server
 - Deployment overview
 - Install Rasa X & Apply domain and SSL
- Code along: Finish up deployment
 - Load assistant into Rasa X
 - Get action server and duckling working
 - Connect to external channel
- Lecture: Introducing Rasa X
 - Why Rasa X
 - How to use Rasa X as your development progresses
- Instructor Demo: Development workflow
 - Review conversations
 - Moving changes to production

Sync your fork with the upstream repository

Pull in the latest changes from upstream. This pulls in a recent update that makes the Financial Demo assistant compatible with Rasa Open Source 1.10.0.

1. Fetch the latest branches and commits from upstream (if you get an error saying no upstream found, run `git remote add upstream https://github.com/RasaHQ/financial-demo.git` and then re-run the fetch command)
 - a. `git fetch upstream`
2. Checkout your local master branch
 - a. `git checkout master`
3. Merge changes from upstream/master into your local master branch
 - a. `git merge upstream/master`
4. Push changes to your remote repository
 - a. `git push origin master`

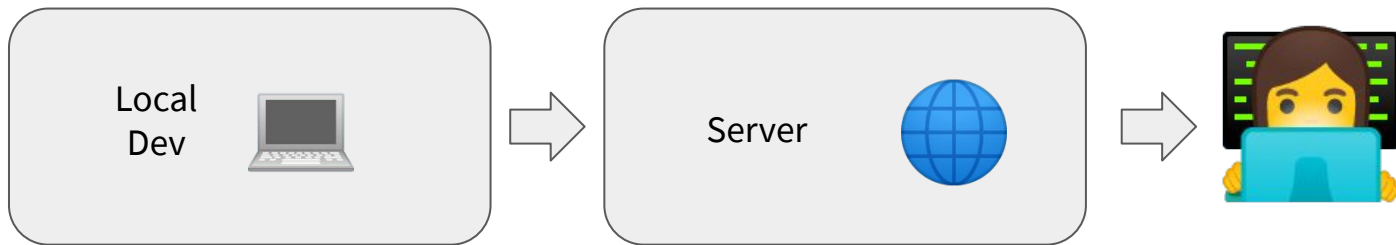
Deployment

Deployment Overview

What do we mean when we say ‘deployment’?

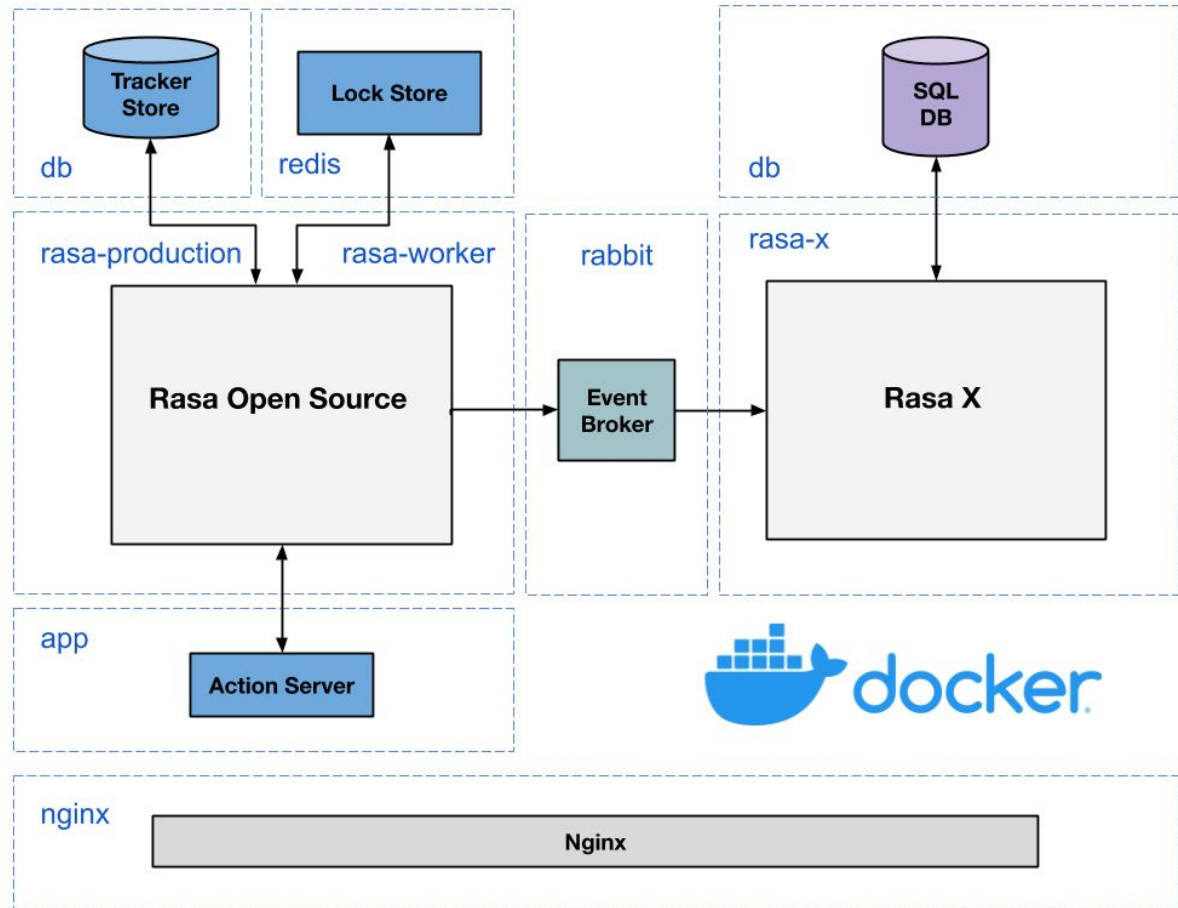
Until now, we’ve been building and testing the assistant on our local machines.

When we deploy, we’ll push the code to a remote server and make the application accessible to users through a live messaging channel integration (Telegram)



Deployment

What are we deploying?



Deployment options

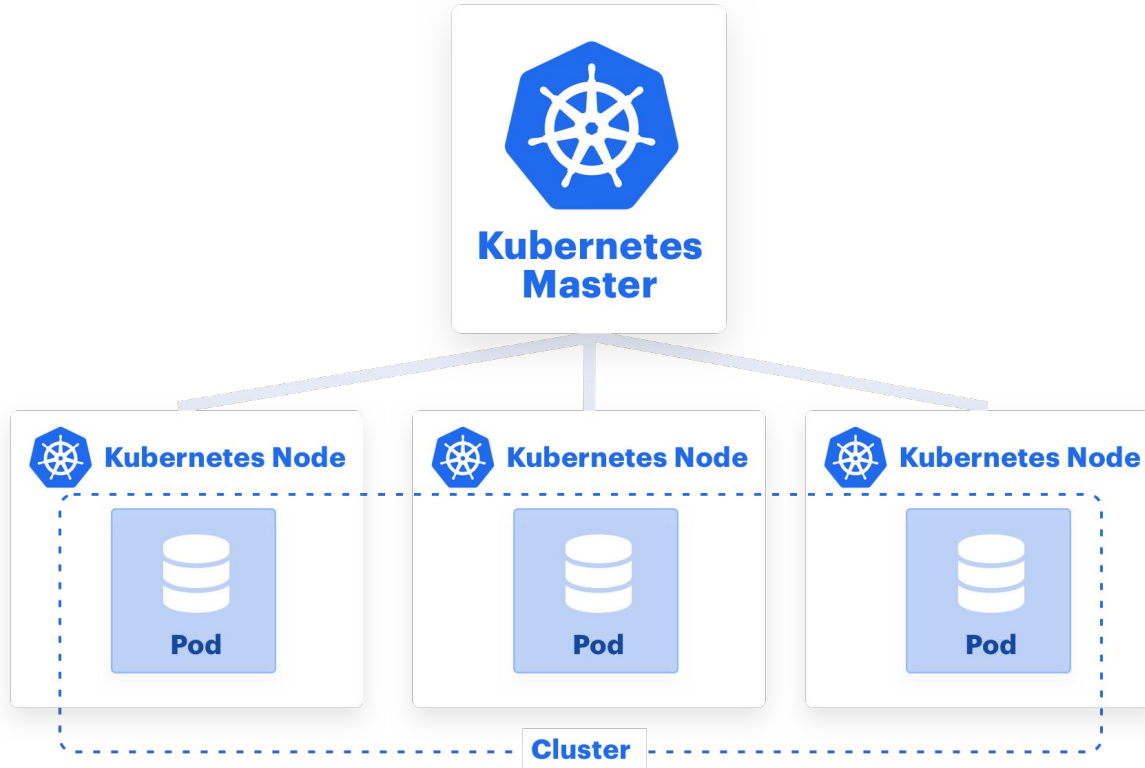
Kubernetes (k8s or k3s)

- Runs multi-container applications on multiple host machines (nodes)
- Pod = smallest deployment unit (can be single or multiple containers).
- Copies of pods are distributed on nodes. The number of copies you want is defined by the deployment.
- If one node goes down, the k8s service creates more pods to maintain the desired number.
- Advantages: resilient, can be configured to auto-scale

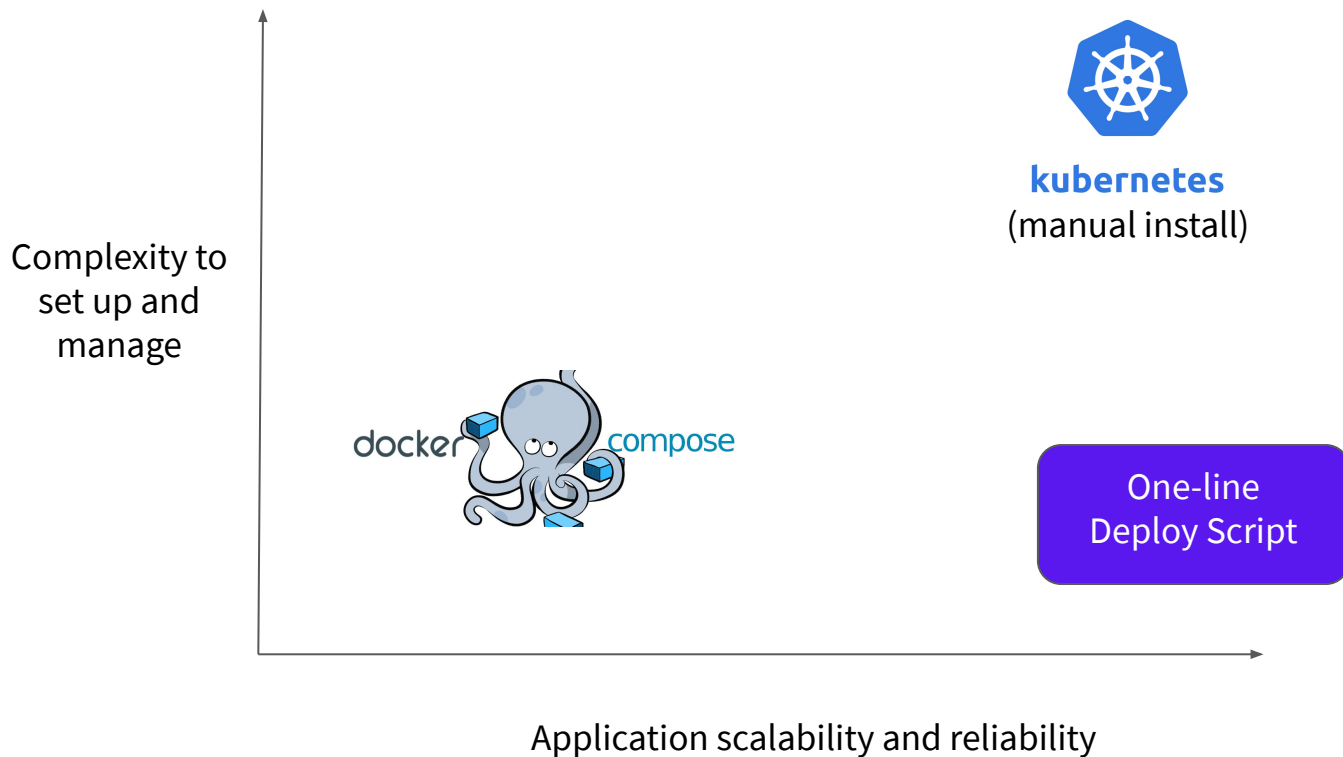
Docker Compose

- Runs multi-container applications on a single host machine
- Networks containers together
- Recommended for development or for smaller deployments

Kubernetes (super high level architecture)



Which deployment option should you choose?




One Line Deploy Script

```
curl -s get-rasa-x.rasa.com | sudo bash
```

Can deploy Rasa Open Source/Rasa X to a Kubernetes cluster you've already created, or it can create one for you. If you use the script to create your cluster, it spins up a k3s cluster. (k3s = lightweight version of Kubernetes)

What the script does:

1. Checks for an existing cluster. If it doesn't find one, it creates one.
2. Installs Helm (Helm = package manager for Kubernetes)
3. Deploys Rasa Open Source and Rasa X using the Rasa X Helm Chart

 Tip: to upgrade your Rasa X version, just re-run the deploy script. It'll automatically upgrade to the latest version

Blog post: [The Complete Guide to Deploying your Rasa Assistant](#)

Demo

In this Demo...

This demo will show you the steps the Rasa team took to set up your server for the workshop. This part won't be interactive (nothing to follow along with here - we've already set this up for you). But feel free to watch and ask questions!

- Create a new virtual machine on Google Cloud platform
- Apply a domain
- Install Rasa Open Source and Rasa X using the one-line deploy script
- Apply an SSL

Integrated Version Control

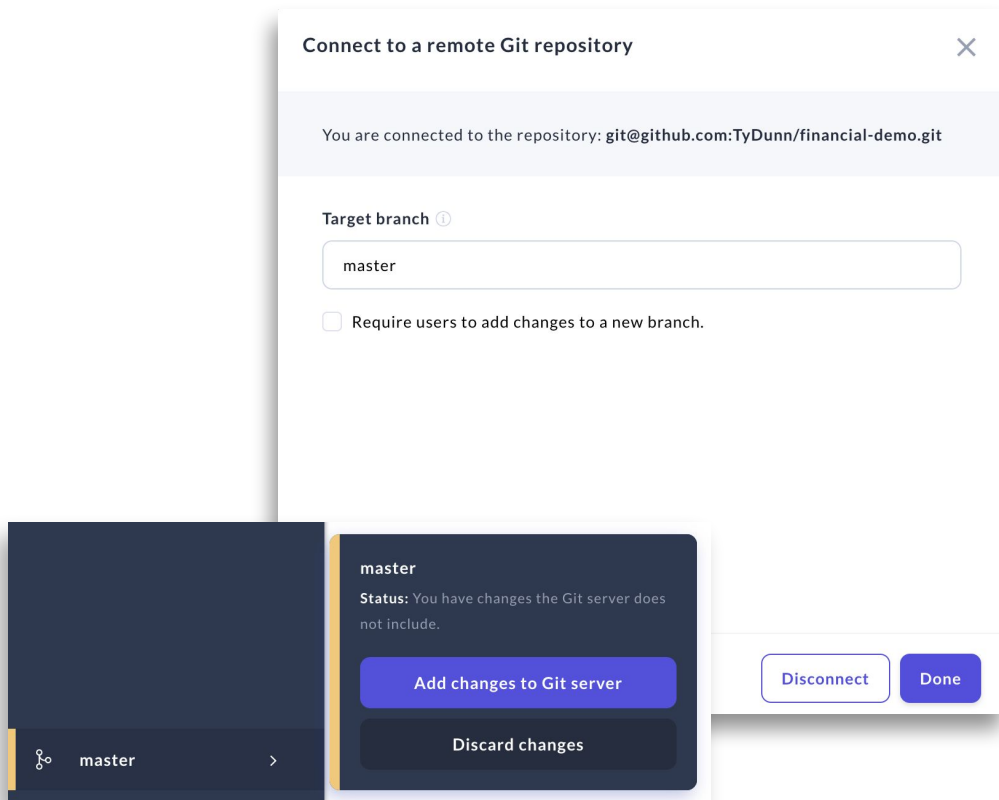
Integrated Version Control

What does it do?

- Creates a two-way sync with a remote git repository (works with any git-based platform: Bitbucket, GitHub, Gitlab, etc.)
- Allows you to version training data and push changes made in Rasa X to a target branch

By extension...

- Connects Rasa X to downstream workflows like automated testing and deployments
- Brings Rasa X in line with software engineering best practices



Connect your assistant to Rasa X

1. Navigate to your forked copy of the Financial Demo assistant
2. Copy the SSH URL from your GitHub repository
3. Connect Integrated Version Control in Rasa X
4. Specify the branch you want to use as **master**
5. Copy and paste the Deploy Key in your GitHub repository settings

Finish Deployment

What's left?

1. Update master branch from upstream repository (RasaHQ/financial-demo)
2. Set up the action server
3. Enable Duckling
4. Connect Telegram

Deploying Custom Actions

When it's deployed, the assistant's custom action server runs in a container.

A **Docker image** is a blueprint for containers that are “spawned” from it. It defines the OS and system requirements, and the dependencies the containerized application needs to run.

The **Dockerfile** contains the instructions for building a Docker image.

Tree: 2d38c5da5f ▾

[financial-demo](#) / Dockerfile



melindaloubser1 update requirements to 1.9

2 contributors



11 lines (7 sloc) | 224 Bytes

```
1 FROM rasa/rasa-sdk:1.9.0
2
3 COPY actions.py /app/actions.py
4 COPY requirements-actions.txt /app
5
6 USER root
7 RUN pip install --no-cache-dir -r requirements-actions.txt
8
9 USER 1001
10 CMD ["start", "--actions", "actions", "--debug"]
11
```

Demo: **Building the Action Server Image**

Steps for Building the Action Server Image

1. Place the Dockerfile in the base of your project directory and note the path to your actions.py file.
2. Run this command in the same directory as your Dockerfile:
 - a. `docker build . -t <account_username>/<repository_name>:<custom_image_tag>`
3. Push the image to the container registry
 - a. Log in (or log in from the Docker desktop app)
 - i. `docker login --username <account_username> --password <account_password>`
 - b. Push the image
 - i. `docker push <account_username>/<repository_name>:<custom_image_tag>`
4. Reference the image in your deployment

Deployment Settings

The one line deploy script surfaces the most common deployment settings as **environment variables** that you can customize. These include variables to set your initial password, connect a messaging channel, install a specific version of Rasa X or Rasa Open Source, and more. We'll use this method to specify the action server image, enable Duckling, and connect Telegram.

To apply the settings, run the export command to export the variable, and then run the one-line deploy script.



You can use this method on a new or existing cluster (e.g. to upgrade to a new version of Rasa X)

```
export INITIAL_USER_PASSWORD="my-safe-password"  
# -E flag applies environment variables from the current user for  
`sudo`  
curl -s get-rasa-x.rasa.com | sudo -E bash
```


Get credentials from Telegram

Before you can connect your assistant to Telegram, you need to register your bot with Telegram and get an API token and username.

Go to [Bot Father](#) on Telegram and type /newbot

Follow the prompts to register your bot and get your API token and username



BotFather

Done! Congratulations on your new bot. You will find it at t.me/RasaFinbot. You can now add a description, about section and profile picture for your bot, see [/help](#) for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

3:



Tip: When registering your bot, the **name** is displayed in contact details. The **username** is a short name used in mentions and must end in bot.

Action Server, Duckling, Telegram

1. Set 5 environment variables:
 - a. `export ACTION_SERVER_IMAGE="karenwhite/dayfourdemo"`
 - b. `export ACTION_SERVER_TAG="version-0.3"`
 - c. `export ENABLE_DUCKLING="True"`
 - d. `export RASA_VERSION="1.10.0"`
 - e. `export ADDITIONAL_CHANNEL_CREDENTIALS="telegram.access_token='<your token>',telegram.verify='username',telegram.webhook_url='https://<yourdomain>.com/webhooks/telegram/webhook/'"`
2. Run the one-line deploy script to apply the environment variables
 - a. `curl -s get-rasa-x.rasa.com | sudo -E bash`

Checking the status of your cluster

<code>kubectl get pods</code>	Lists the pods running in the k8s cluster, along with status
<code>kubectl logs <service name></code> example: <code>kubectl logs rasa-app-55b866cd99-k6nqp</code>	List the logs for a single pod, for example, the action server

Finish Deployment




Talk to your bot on Telegram!

Visit https://web.telegram.org/#/im?p=@your_bot_name and test your new messaging channel!

Rasa X Tour

Rasa X turns conversations into training data



Review  Annotate  Improve 

Real Conversations > Synthetic Data

During development, training data is created by:

- Writing down hypothetical conversations and training examples
- Studying existing chat transcripts or knowledge bases
- Consulting subject matter experts

That's a great start, but all of those scenarios are based on human:human conversations.

Human:bot conversations have different patterns - users say different things when they are talking to a bot.

The best training data is generated from the assistant's
actual conversations

Conversation Driven Development (CDD)

The challenge

When developing assistants, it's impossible to anticipate all of the things your users might say.

The approach

A user-centric process: listening to your users and using those insights to improve your AI assistant.

Principles

Prototype, Read, Annotate, Test, Track, Fix

Blog post: [Conversation-Driven Development](#)

Yesterday we shared a link to test the Financial Demo Bot....

Let's see how the assistant did, and walk through how to review conversations in Rasa X!

GIVE TO USERS ASAP

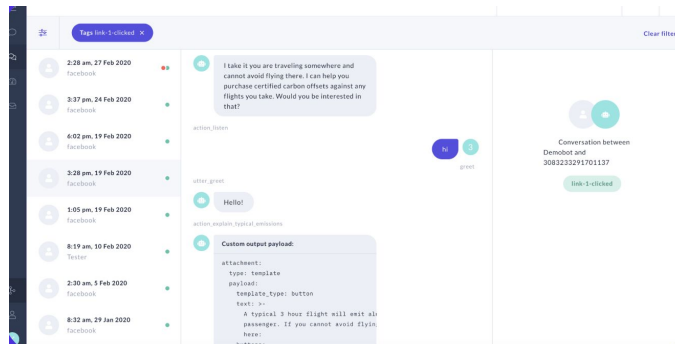
Wow. It struggled. Assistants always do in the beginning

- You can't anticipate how users will interact with your assistant
- Expect your assistant to make mistakes when you first share it with users, no matter how much you design
- However, you can improve your ability to expect the unexpected by learning from real data
- Take home message: **share your assistant with users as soon as possible**

REVIEW CONVERSATIONS

Drive your development based on feedback from users

1. Track user actions and success rates using tags
 - a. Tags can be applied via API or in the UI
2. Review conversations and see where they fail
3. Use feedback to adjust flows and response templates, improve accuracy, or prioritize feature requests



- ☐ docs search unhelpful
- ☐ negative feedback
- ☐ positive feedback
- ☐ docs search helpful

Development workflow

How Rasa X fits into your development workflow

Through the Rasa X UI, you can edit certain files:

nlu.md - add and annotate new training examples and intents, map synonyms

stories.md - add new stories to training data

domain.yml - edit file, add/edit response templates

config.yml - edit pipeline configuration

Rasa X isn't meant to replace development that you do locally. For example, custom actions aren't exposed in Rasa X and should be updated in your text editor.

What types of changes should you make in Rasa X vs developing locally?

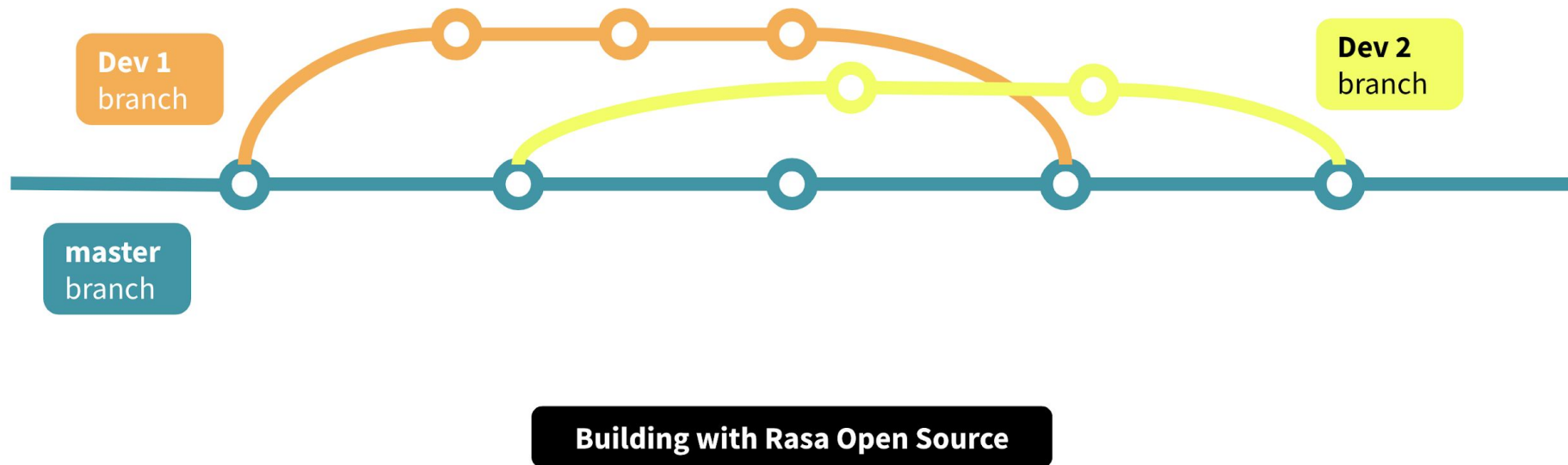
Rasa X

- Everyday training data maintenance:
 - Reviewing conversations
 - Adding, annotating, correcting training examples
 - Adding new test cases
- Edit response templates
- Tagging and sorting conversations

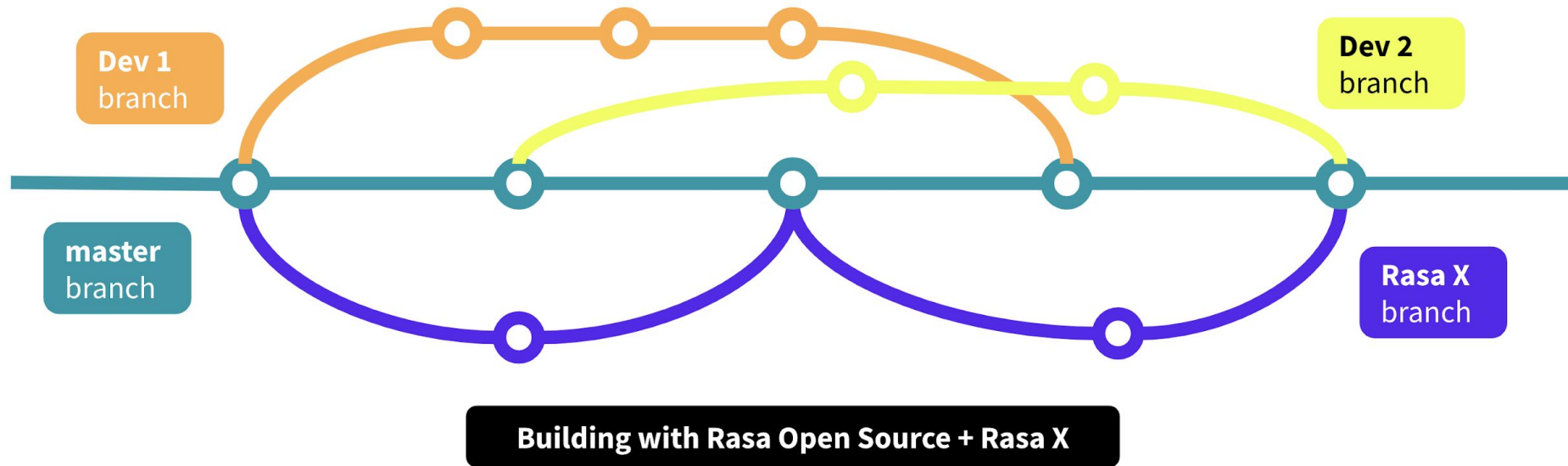
Local Text Editor

- Bigger changes, like adding a new intent
- Updates to custom actions, like changing a form's behavior

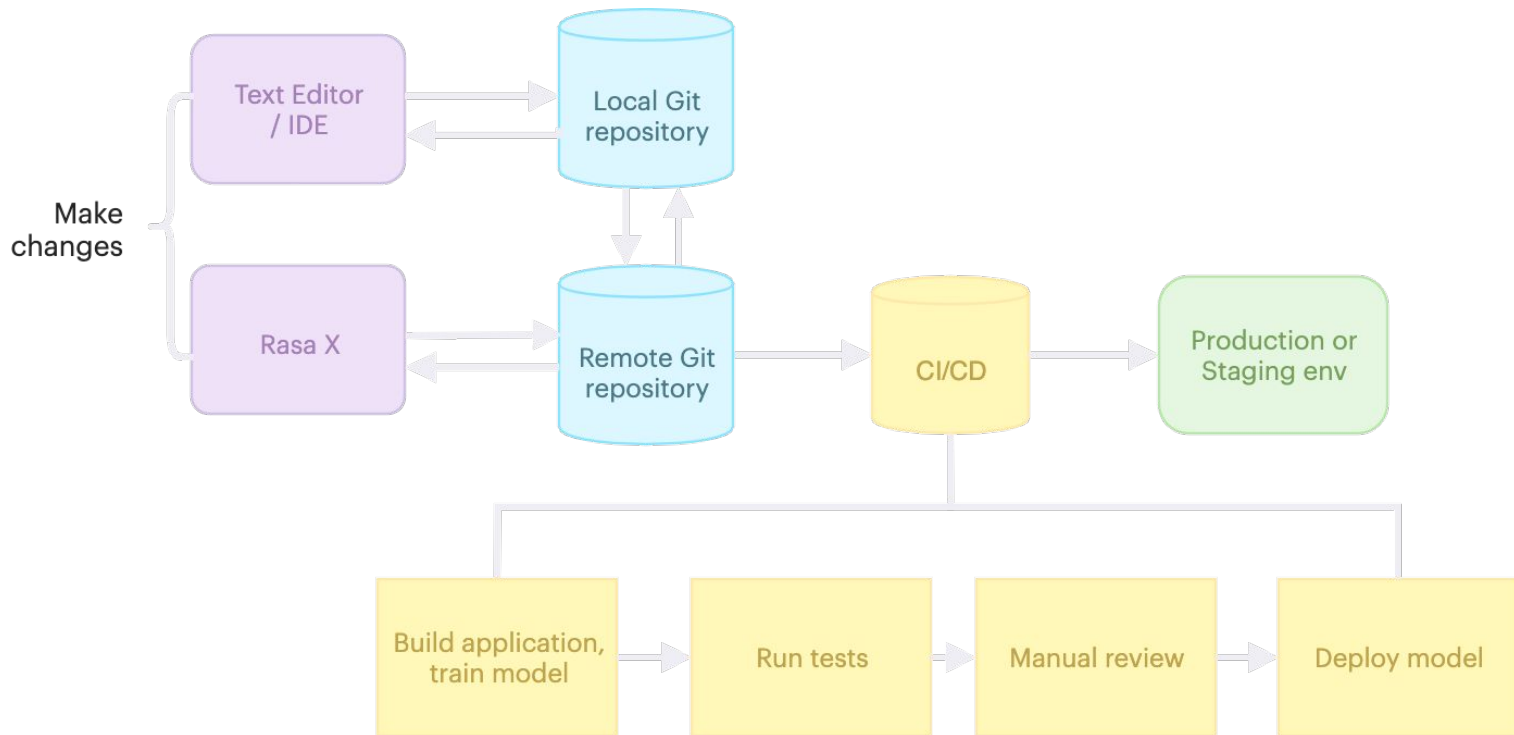
Rasa X is like another development branch....



Rasa X is like another development branch....



End-to-end workflow for managing assistant updates



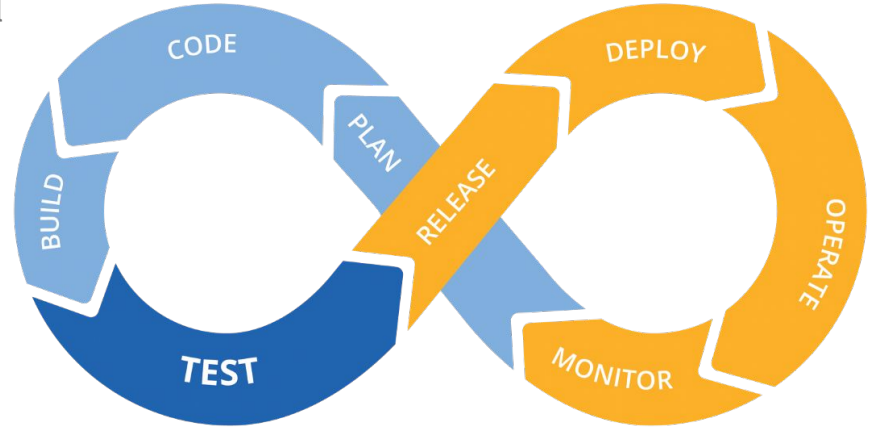
Continuous Integration/Continuous Deployment

What?

Instead of pushing periodic, major releases, CI/CD systems encourage small updates to be released to production all the time. The process of moving code from version control (GitHub) to production is fully or partially automated and tests are often run along the way.

Why?

Small releases reduce the risk of introducing bugs, because it's easier to untangle potential problems. CI/CD workflows also get updates into users' hands faster, resulting in a shorter feedback-to-product cycle.



[Image Credit](#)

CI/CD lets you automate testing for your assistant

```
53 - name: Rasa Data Validation
54   working-directory: ${github.workspace}
55   run: |
56     rasa data validate --debug
57 training-testing:
58   name: Testing Stories
59   runs-on: ubuntu-latest
60   needs: [data-validation]
61   steps:
62   - uses: actions/checkout@v1
63   - name: Set up Python 3.7
64     uses: actions/setup-python@v1
65     with:
66       python-version: 3.7
67   - name: Install dependencies
68     run: |
69       python -m pip install --upgrade "pip<20"
70       pip install -r requirements-dev.txt
71   - name: Cross-validate NLU model
72     run: |
73       rasa test nlu -f 3 --cross-validation
74       python .github/workflows/format_results.py
75   - name: post cross-val results to PR
76     uses: amn41/comment-on-pr@comment-file-contents
77     continue-on-error: true
78   env:
79     GITHUB_TOKEN: ${secrets.GITHUB_TOKEN}
80   with:
81     msg: results.md
```

Validate data to check for mistakes

Test NLU model

Print results to a pull request

[Link to full example](#)

DEVELOPMENT WORKFLOW

After testing, CI/CD lets you automate the process of releasing to production

```
7 jobs:
8   docker:
9     name: Build Action Server Docker Image
10    runs-on: ubuntu-latest
11
12   env:
13     DOCKERHUB_USERNAME: oakela
14
15   steps:
16     - name: Checkout git repository 📄
17       uses: actions/checkout@v2
18
19     - name: Login to DockerHub Registry 🔑
20       run: echo "${{ secrets.DOCKERHUB_PASSWORD }}" | docker login -u ${ env.DOCKERHUB_USERNAME } --password-stdin || true
21
22     - name: Pull latest${{ matrix.image.tag_ext }} Docker image for caching
23       run: docker pull rasa/financial-demo:latest || true
24
25     - name: Build latest${{ matrix.image.tag_ext }} Docker image
26       run: docker build . --tag rasa/financial-demo:latest --cache-from rasa/financial-demo:latest
27
28     - name: Push image with latest tag 📦
29       if: github.event_name == 'push' && github.ref == 'refs/heads/master'
30       run: docker push rasa/financial-demo:latest
```

Build and push latest version of Action Server image to DockerHub

[Link to full example](#)

Automatically upload the tested model and make it active

```
# Upload model
cd model; export MODELNAME=$(ls); curl -k --fail -H "Authorization: Bearer ${RASA_X_TOKEN}" -F "model=@${MODELNAME}" https://carbon
# ensure model is ready and tag as production
sleep 5
ls | xargs -I{} basename {} .tar.gz
export MODEL=$(ls | xargs -I{} basename {} .tar.gz); curl --fail -XPUT -H "Authorization: Bearer ${RASA_X_TOKEN}" https://carbon.ras
```

[Link to full example](#)

cURL call to Rasa X API to upload model

Use Rasa X to build an end-to-end test set from real conversations

The screenshot displays the Rasa X interface, which is used for building and testing chatbots. It is divided into two main panels. The left panel shows a conversation history with a light blue background. It includes user utterances in white bubbles and bot responses in blue bubbles. Bot actions are listed on the right side of the conversation. The right panel shows the 'Story till now' section, where the current conversation is being converted into an end-to-end test. It features tabs for 'Story' and 'End-to-end Story', with the latter being the active tab. The end-to-end test is displayed as a YAML snippet, and a 'Save end-to-end test' button is located at the bottom right of this panel.

Conversation History (Left Panel):

- Bot action: `action_listen`
- User utterance: `utter_greet` (Hi, how can I help you?)
- Bot action: `action_listen`
- User utterance: `action_listen` (whats my balance)
- Bot action: `check_balance` (payment_amount: "balance")
- Slot: `slot` (payment_amount: "balance")
- Bot action: `action_account_balance`
- User utterance: `action_listen` (Your balance is \$1000)
- Bot action: `action_listen`

End-to-end Story (Right Panel):

```
## Story from conversation with 8859bea5-982'  
* greet: hi  
- utter_greet  
* check_balance: whats my [balance](payment_ar  
- slot("payment_amount": "balance")  
- action_account_balance
```

Buttons: `Save end-to-end test`

- Turn real conversations into end-to-end tests using Rasa X
- Collect a sample of the true distribution of real conversations
- Run them as part of an automated testing pipeline

Development Stages

THE APPROACH

Continually improve your assistant using Rasa X

Ensure your new assistant passes tests using **continuous integration (CI)** and redeploy it to users using **continuous deployment (CD)**

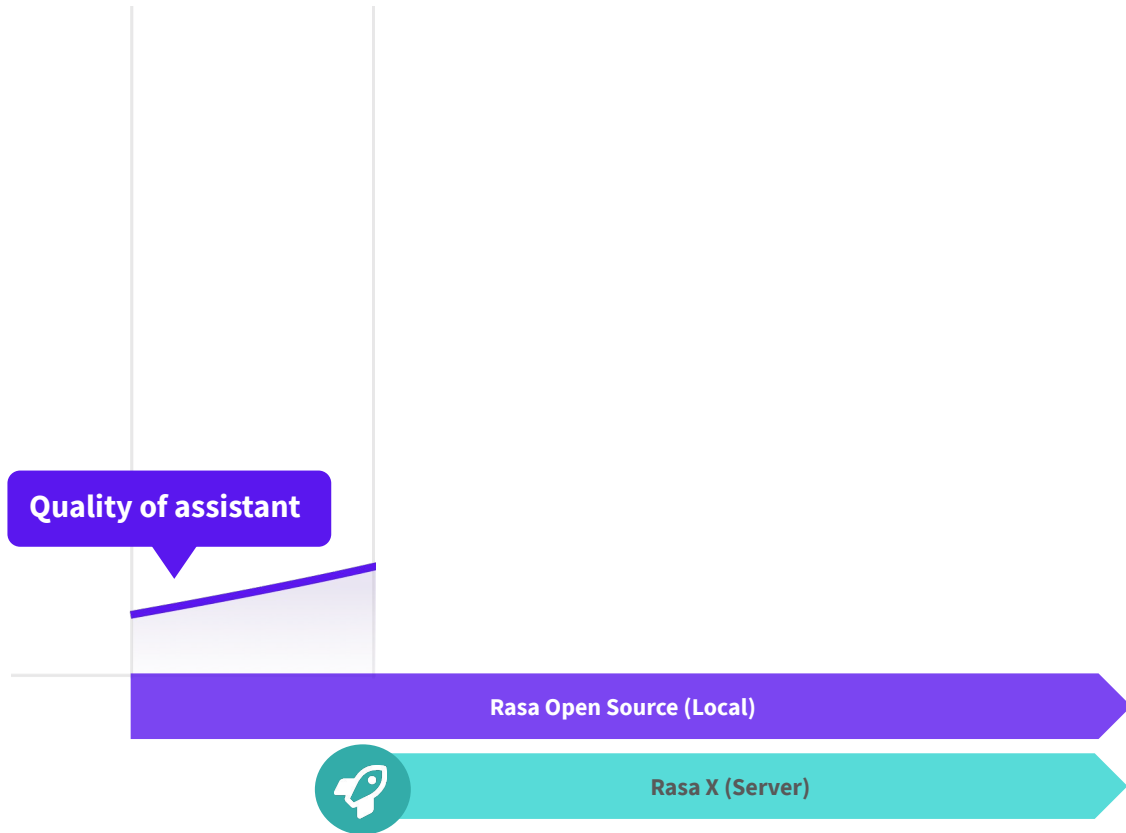


Collect conversations between users and your assistant

Review conversations and **improve your assistant** based on what you learn

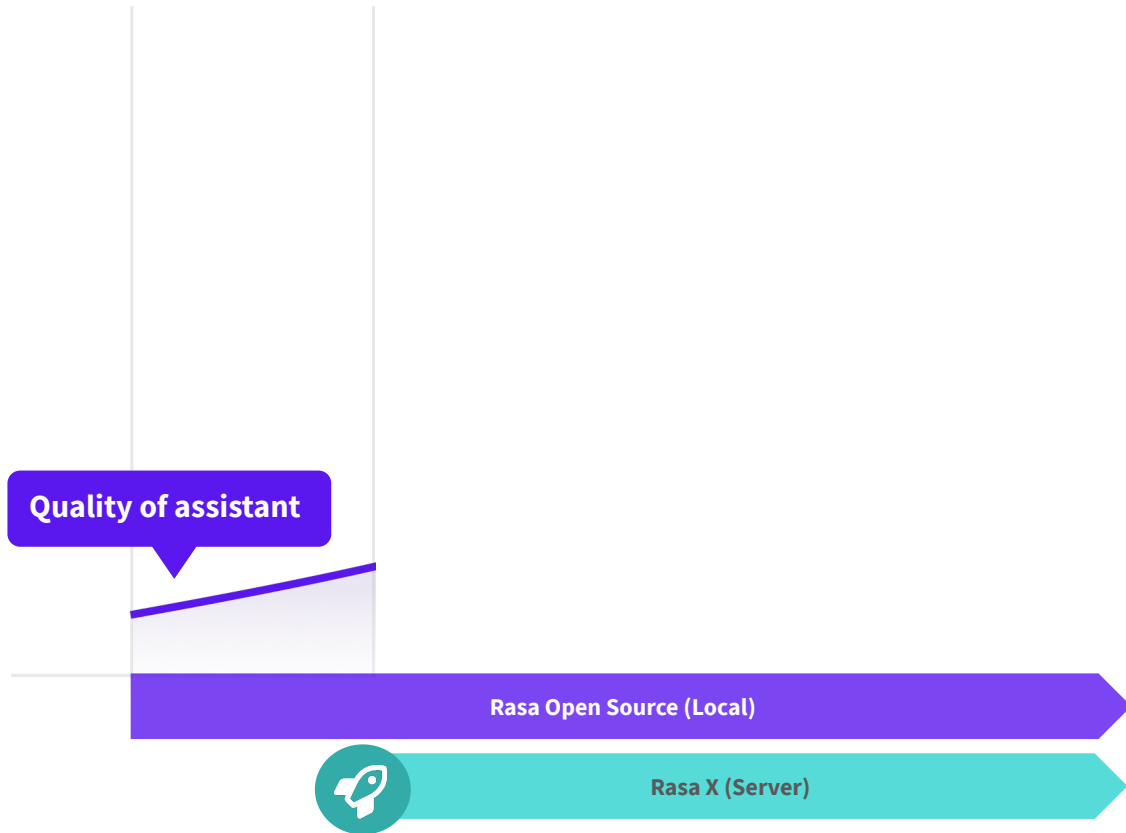
THE APPROACH

The path to a contextual assistant



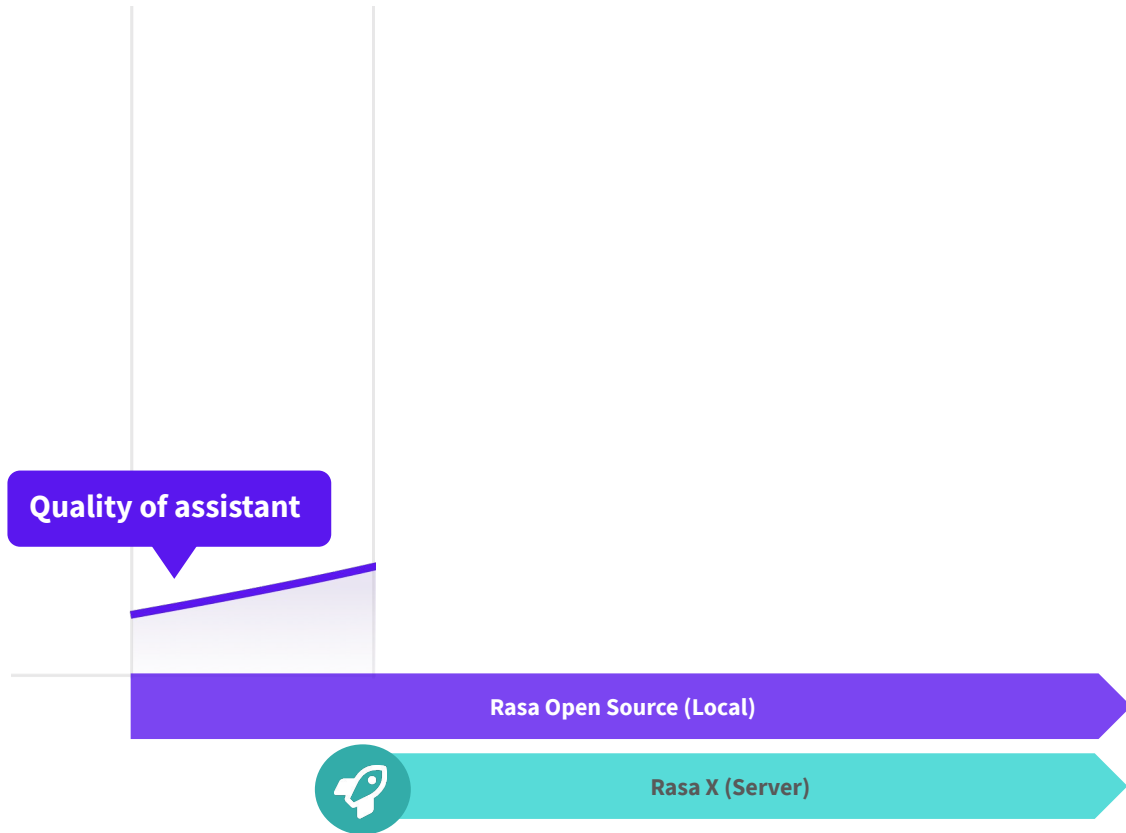
THE APPROACH

The path to a contextual assistant



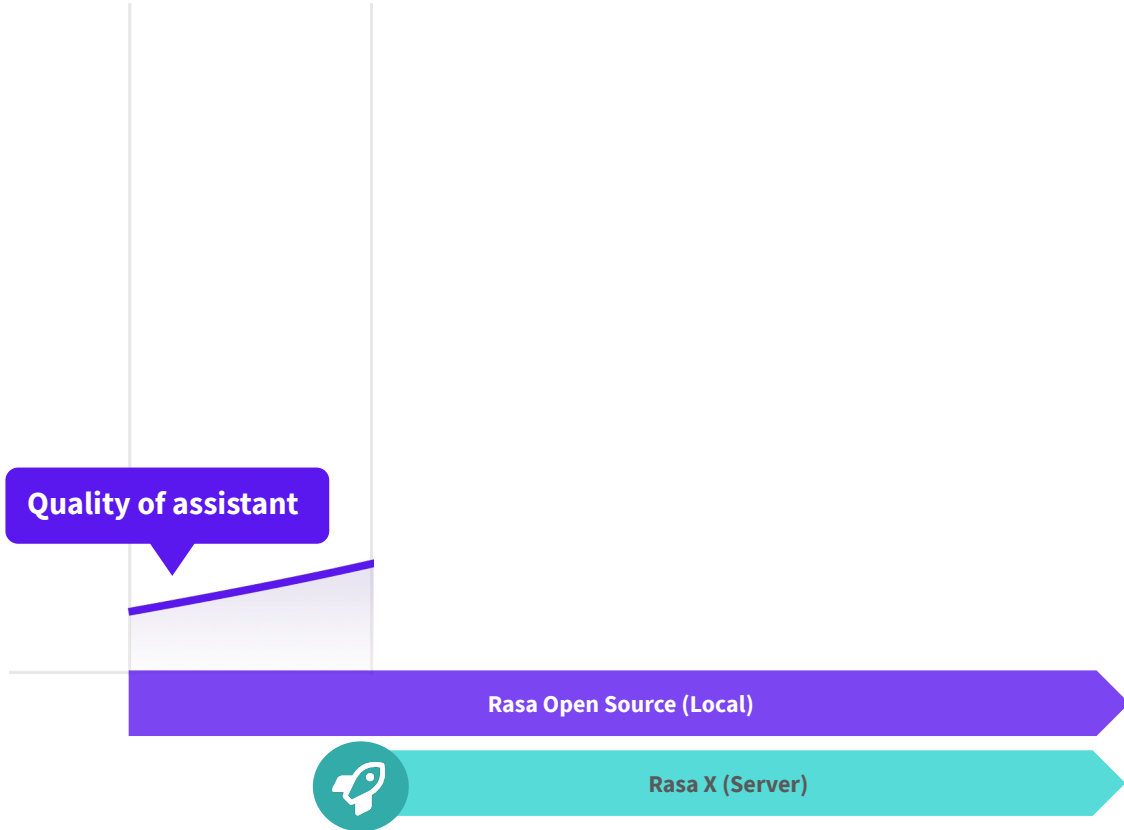
THE APPROACH

The path to a contextual assistant



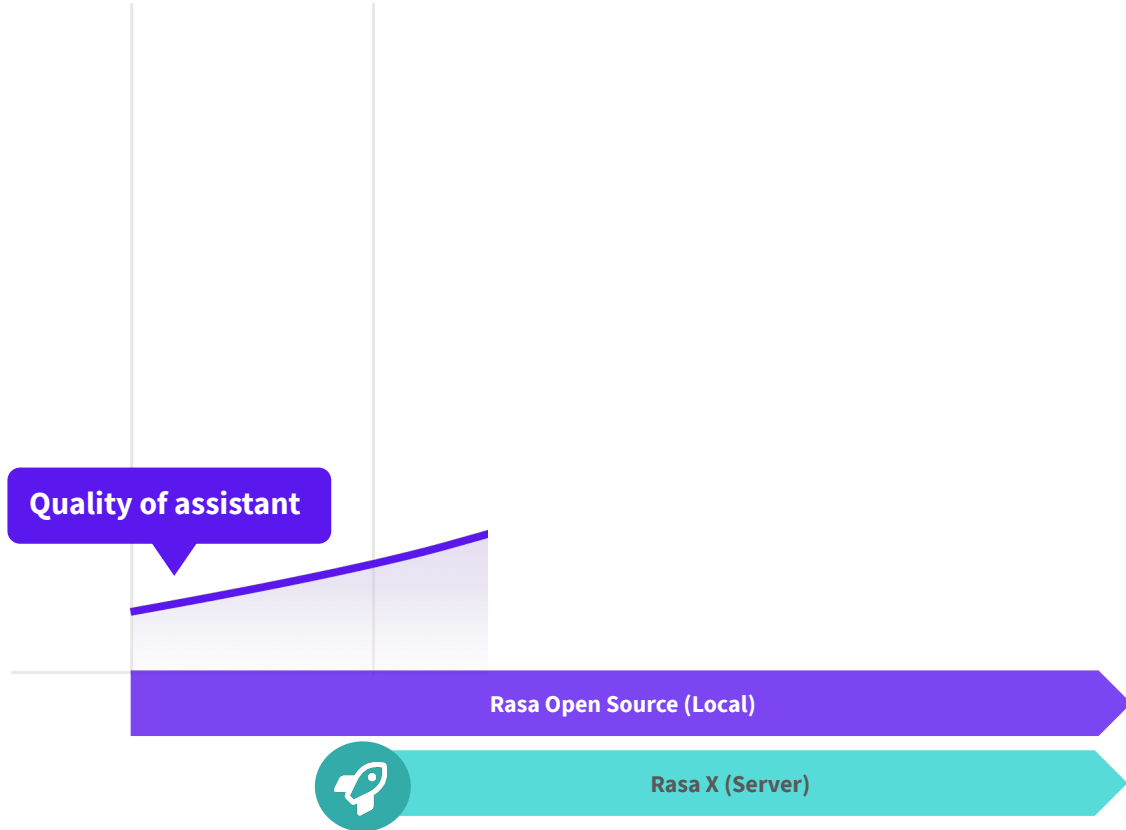
THE APPROACH

The path to a contextual assistant



THE APPROACH

The path to a contextual assistant

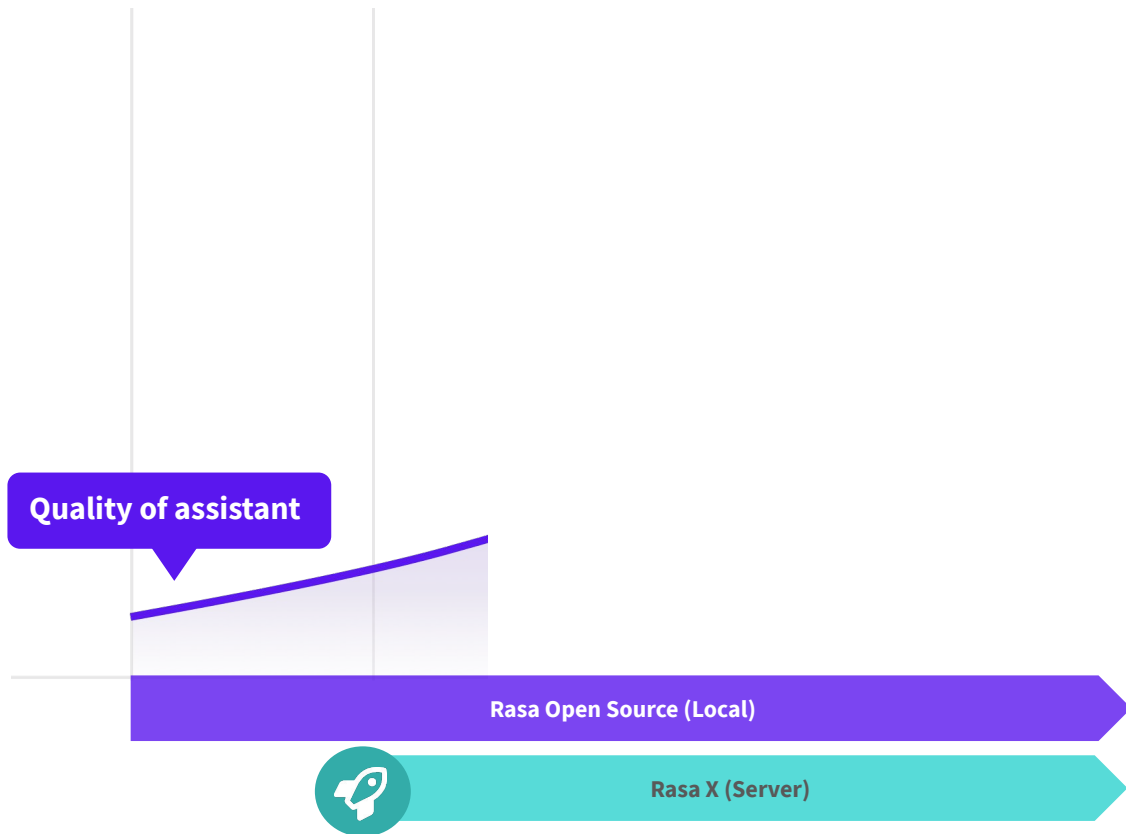


The path to a contextual assistant



THE APPROACH

The path to a contextual assistant

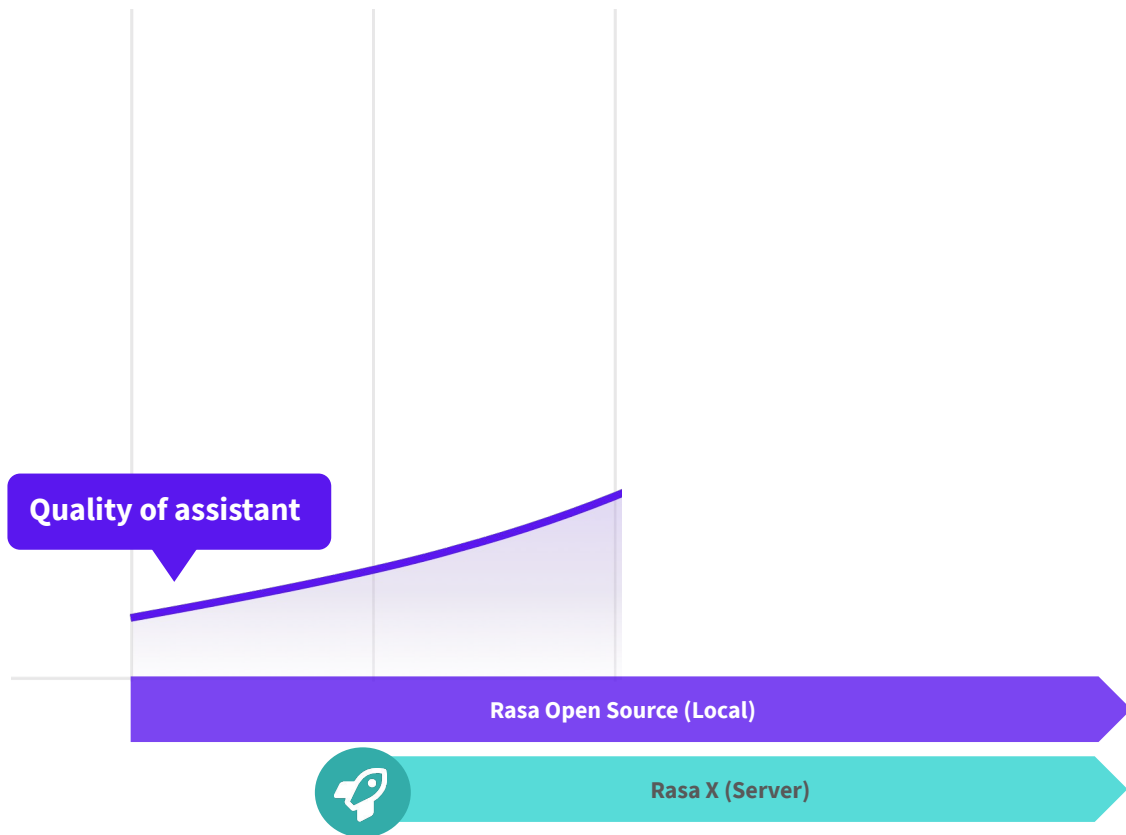


The path to a contextual assistant



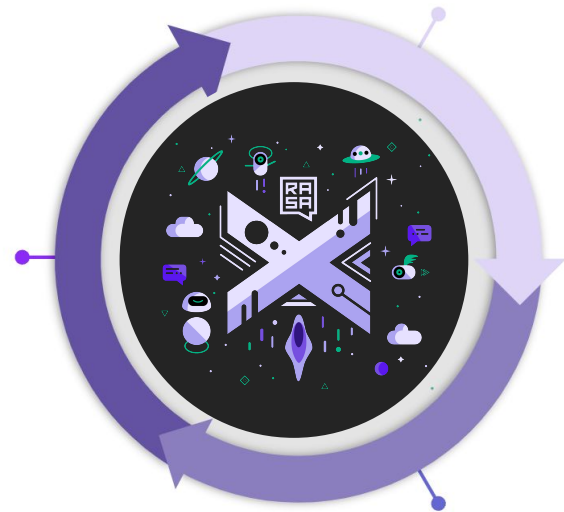
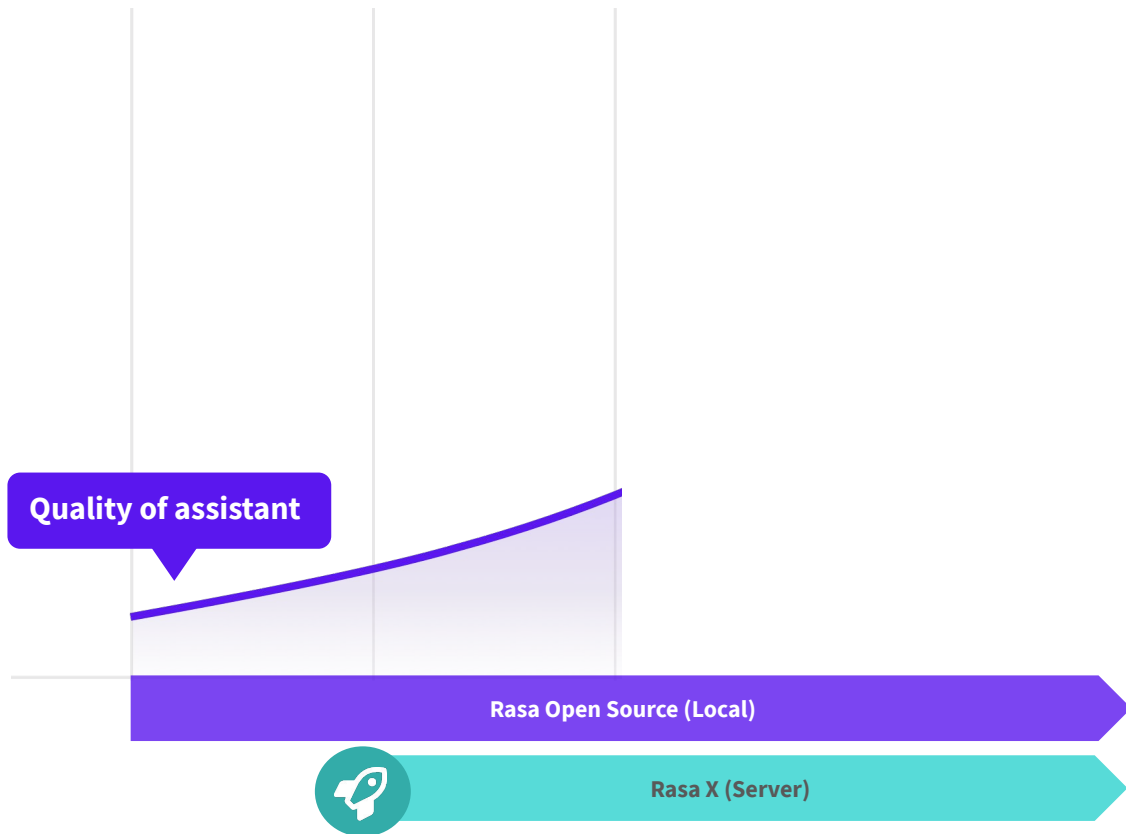
THE APPROACH

The path to a contextual assistant



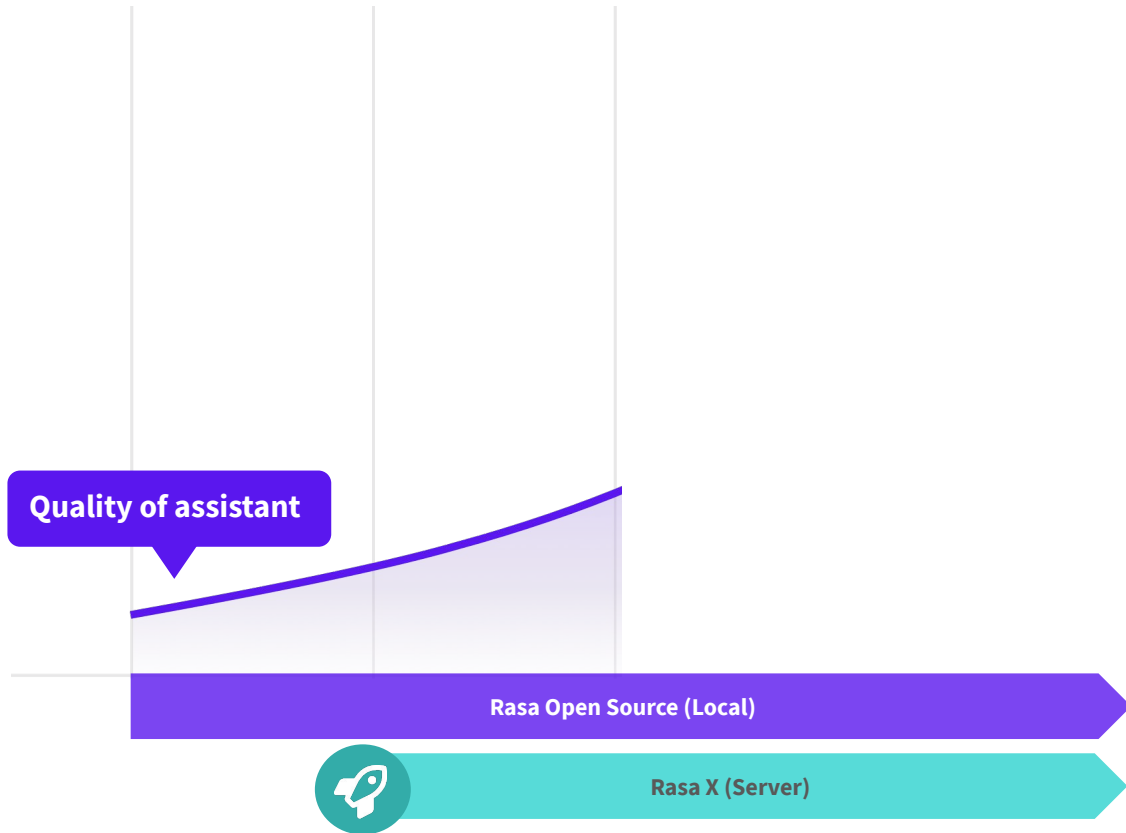
THE APPROACH

The path to a contextual assistant



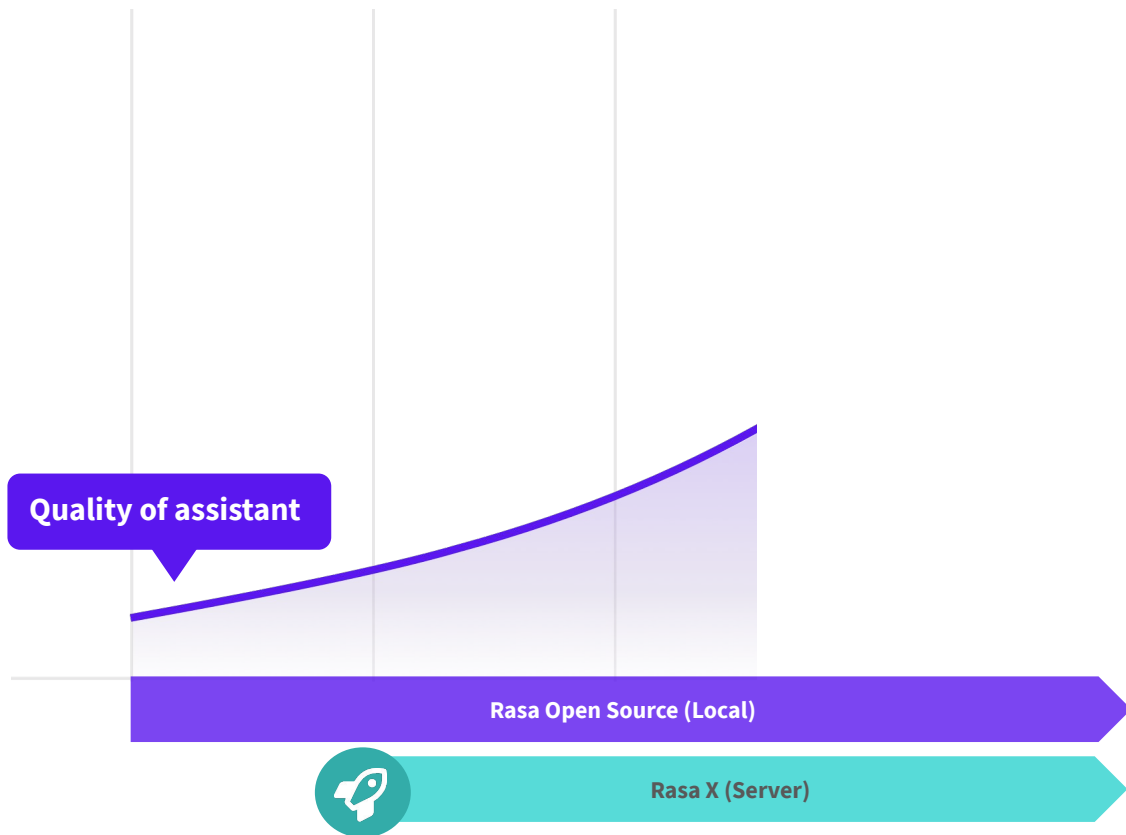
THE APPROACH

The path to a contextual assistant



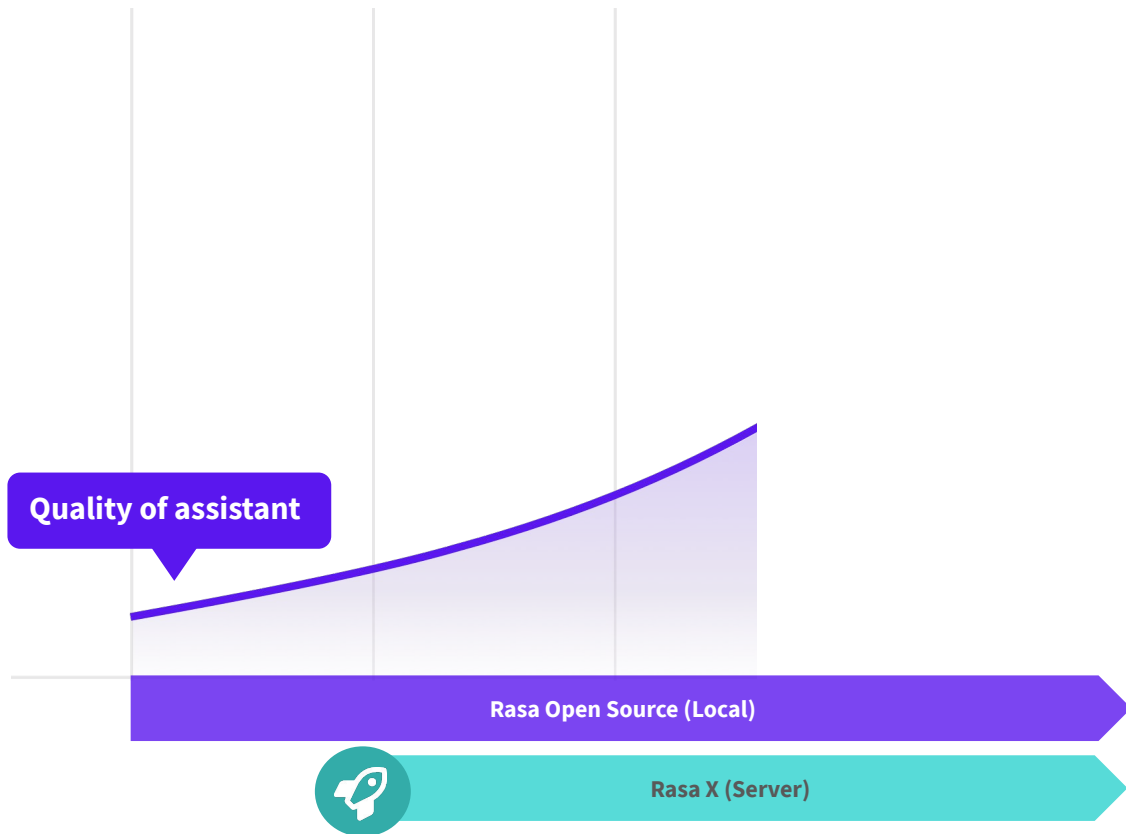
THE APPROACH

The path to a contextual assistant



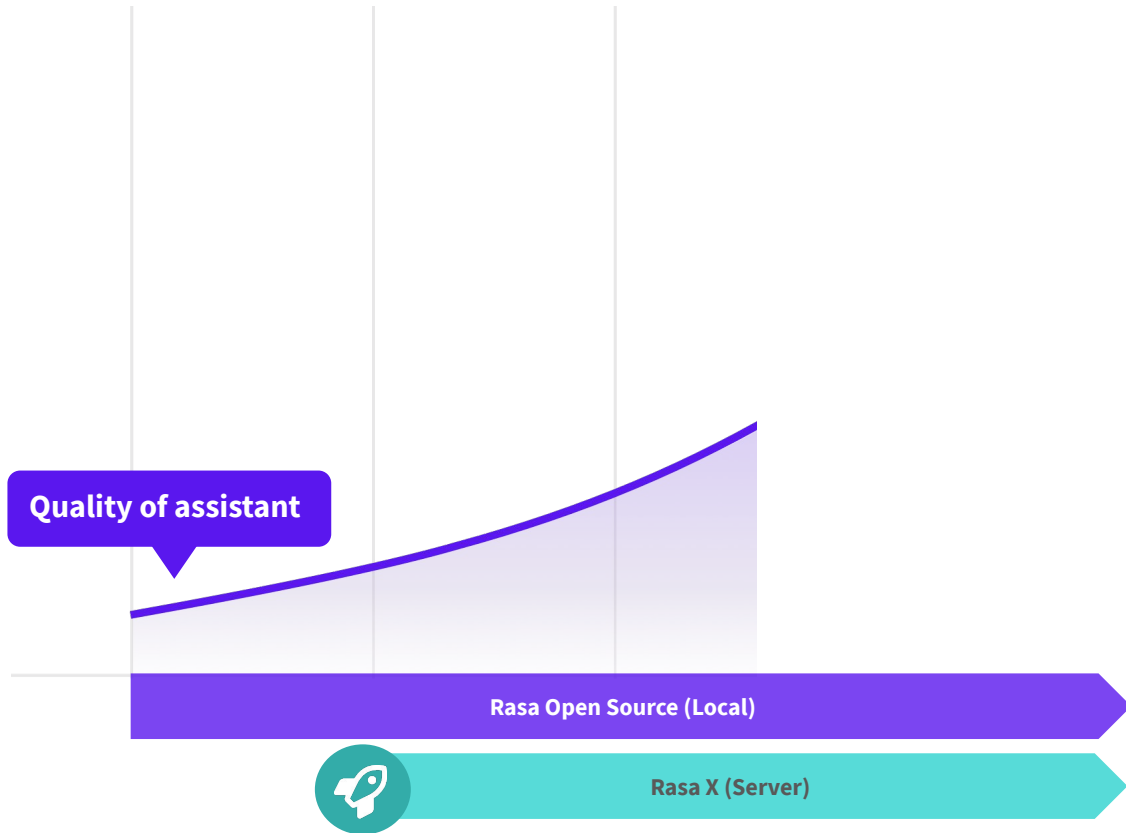
THE APPROACH

The path to a contextual assistant



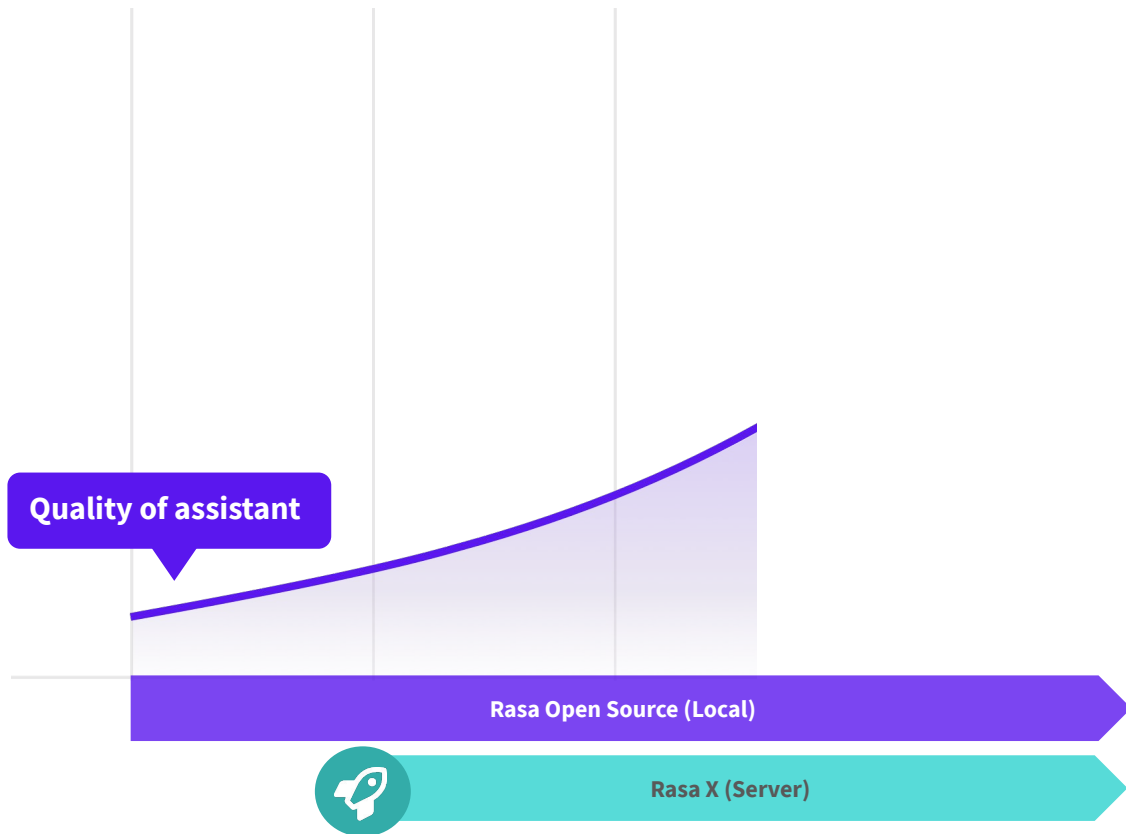
THE APPROACH

The path to a contextual assistant



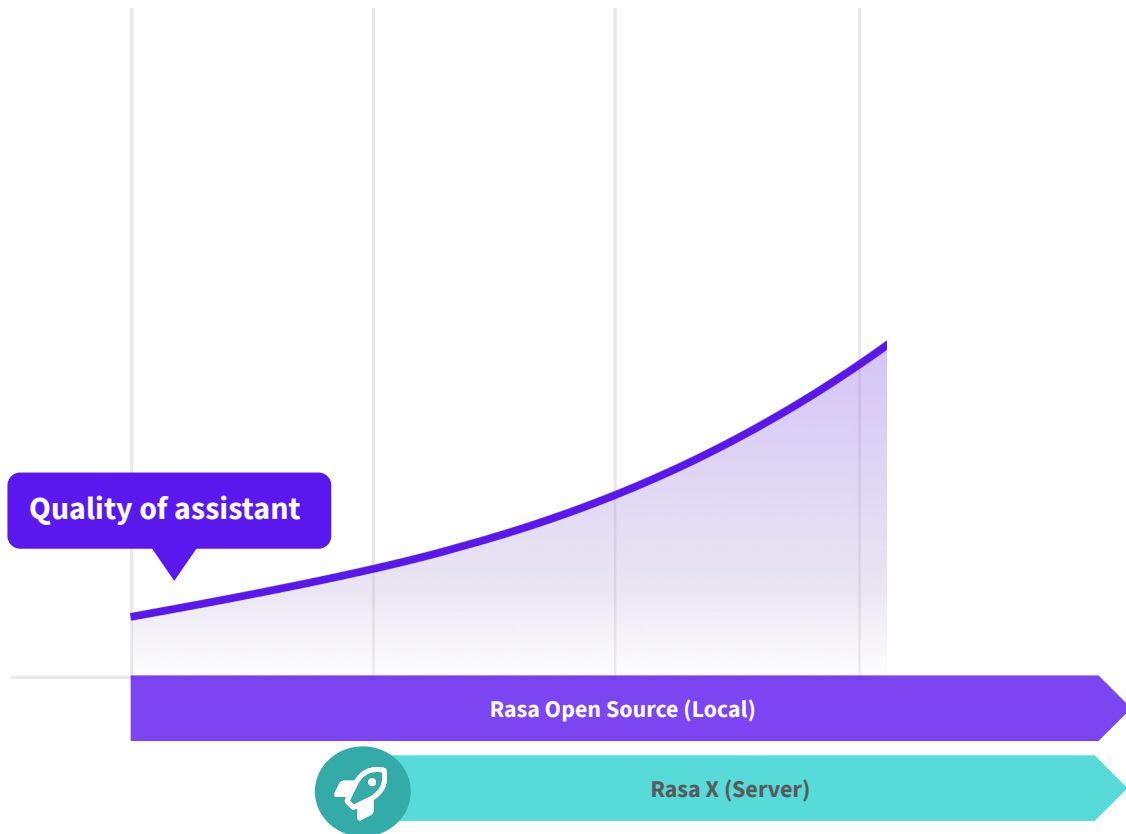
THE APPROACH

The path to a contextual assistant



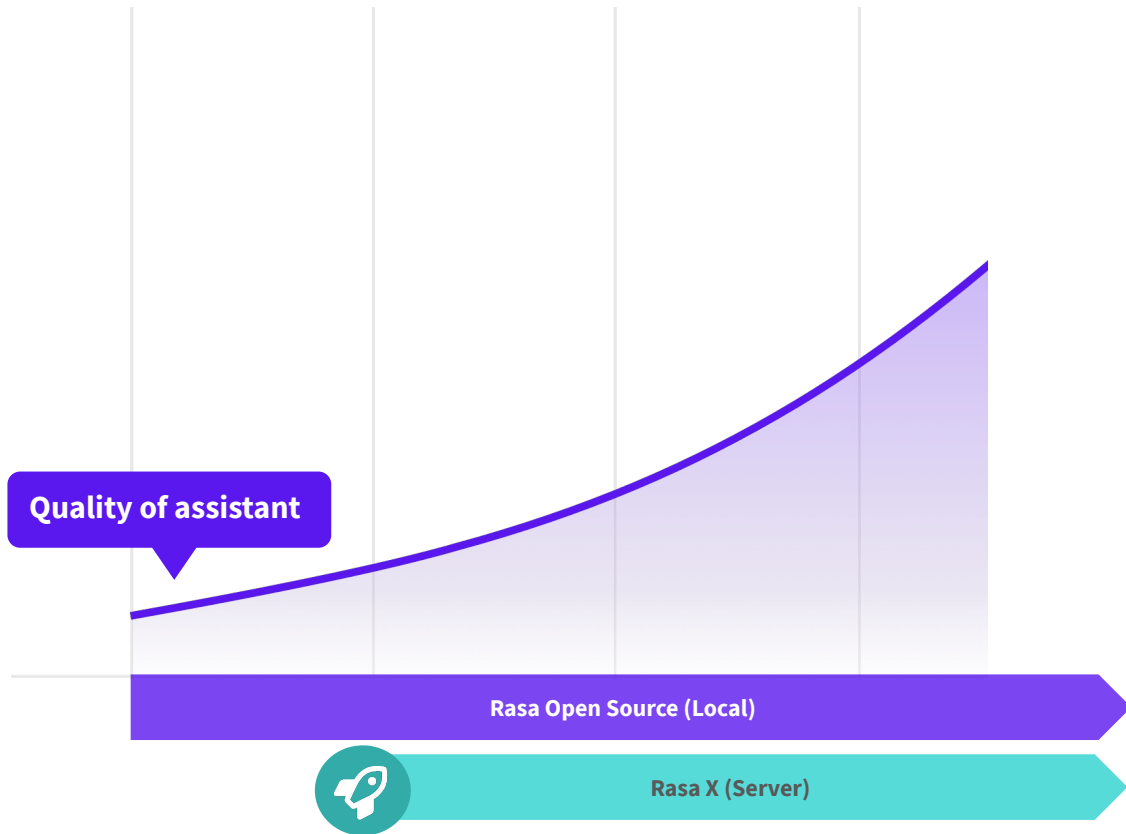
THE APPROACH

The path to a contextual assistant



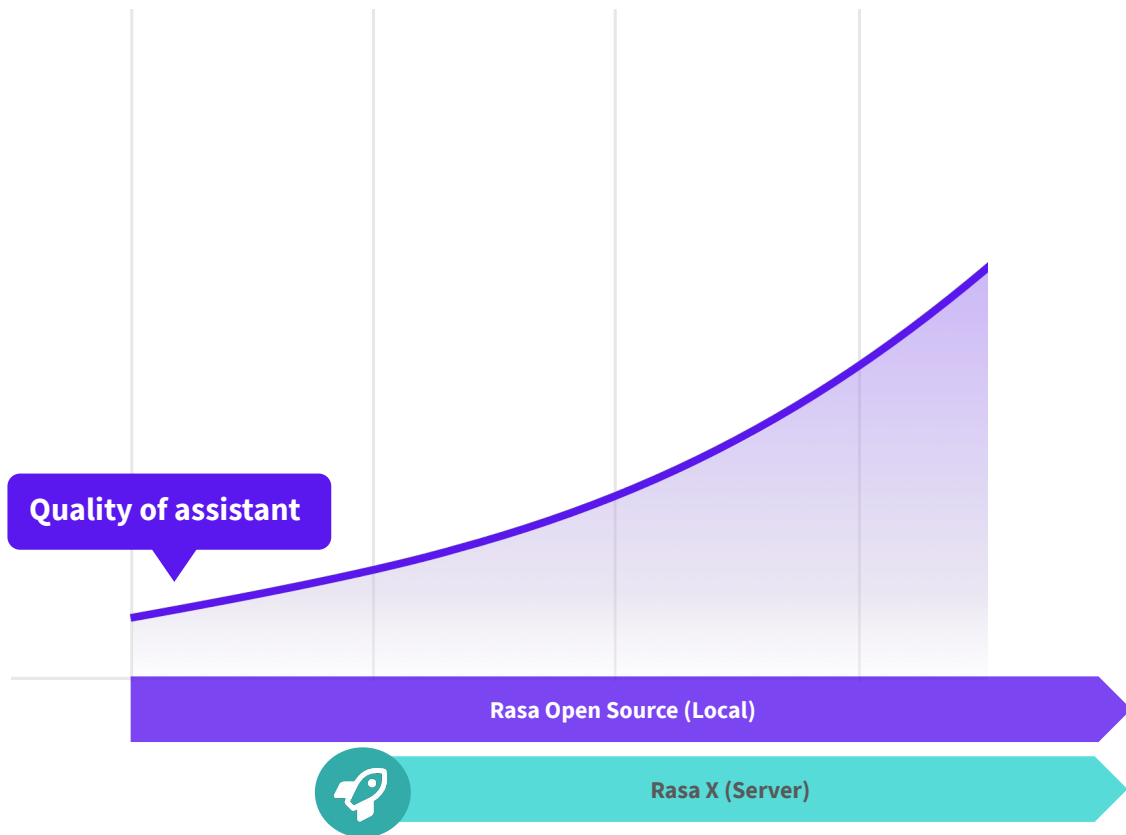
THE APPROACH

The path to a contextual assistant

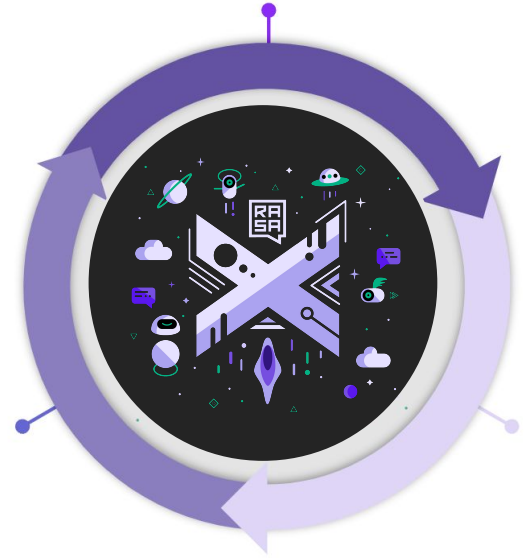


THE APPROACH

The path to a contextual assistant

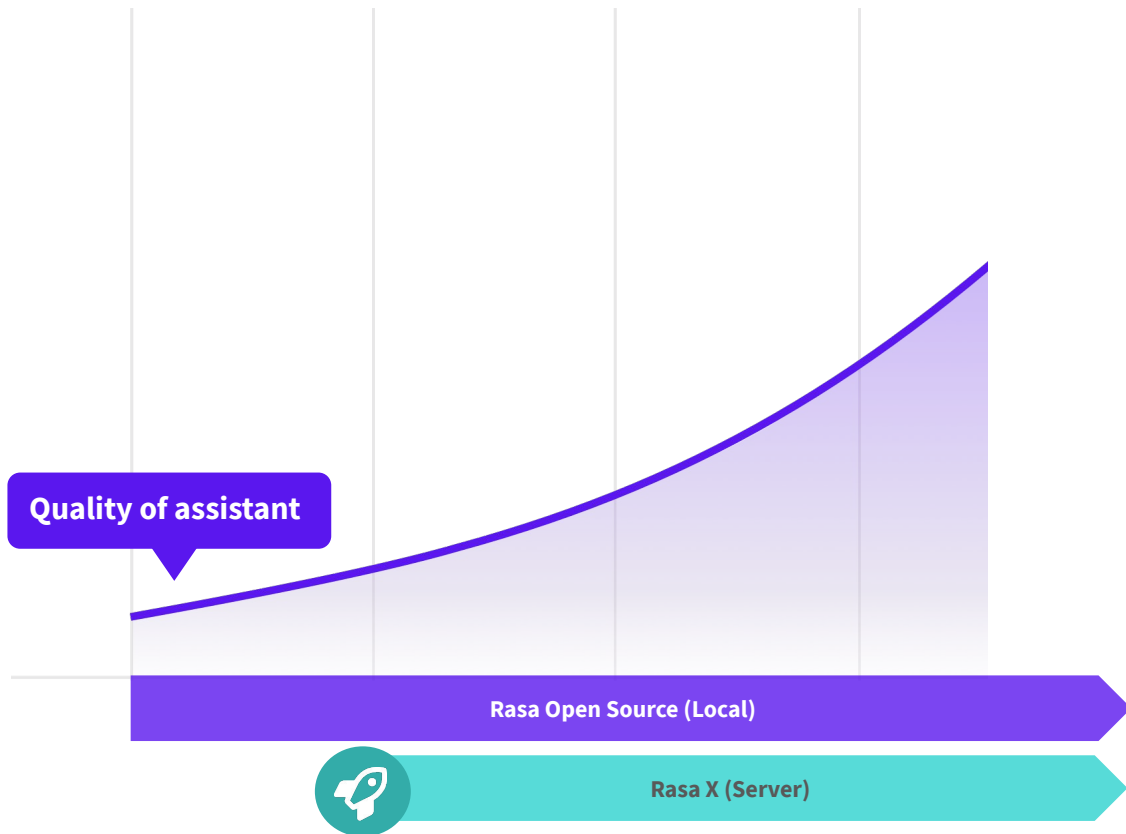


The path to a contextual assistant



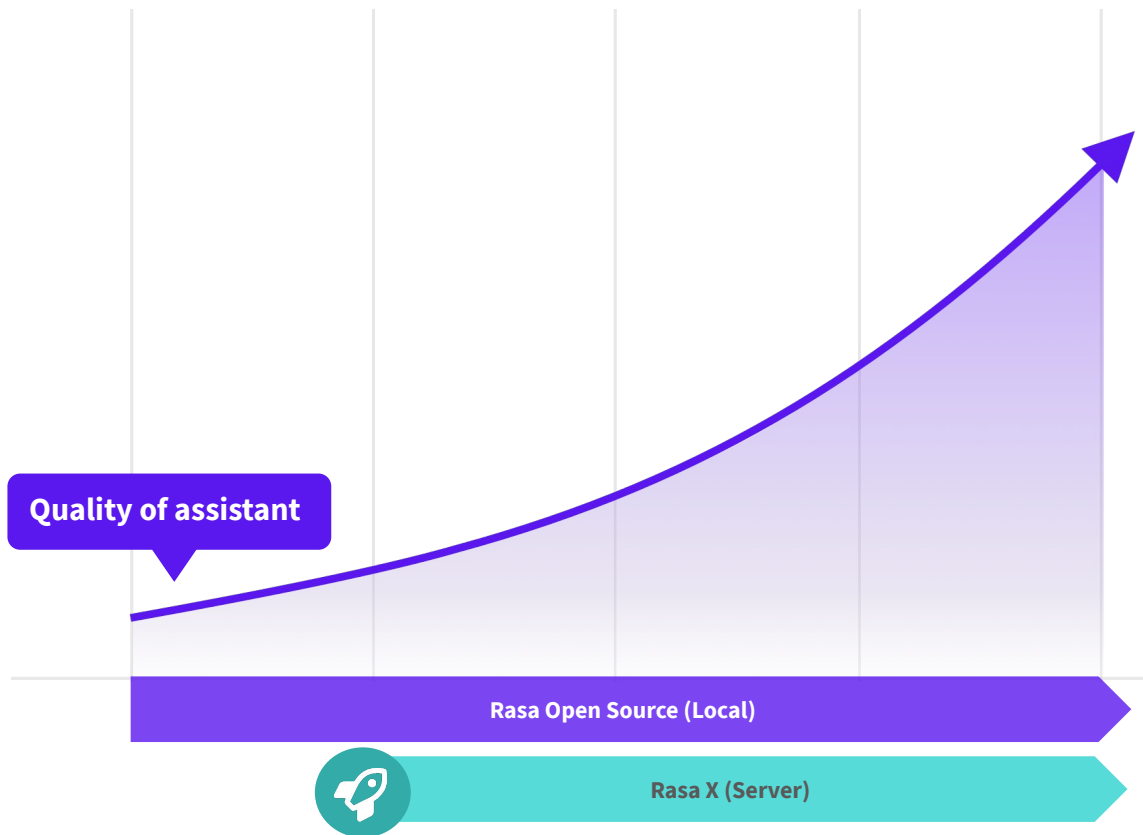
THE APPROACH

The path to a contextual assistant



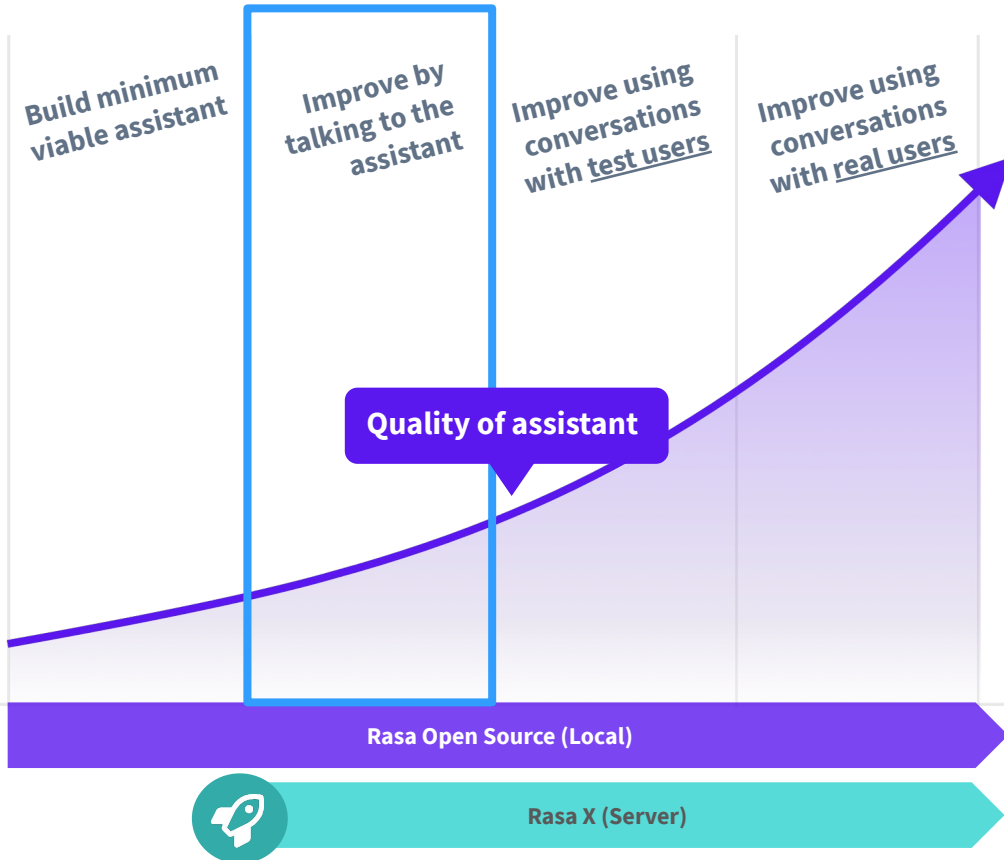
THE APPROACH

The path to a contextual assistant



Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X

Talk to
your bot

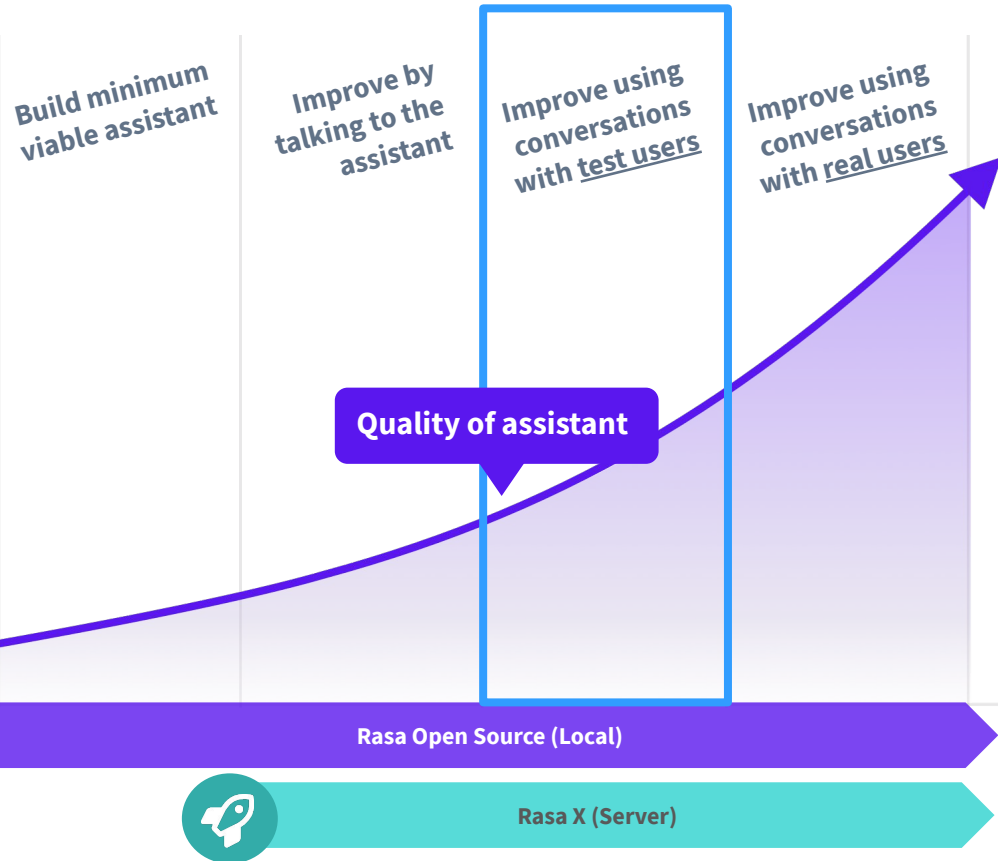


Rasa Open Source is an open source framework for natural language understanding, dialogue management, and integrations.

Rasa X is a toolset used to improve a contextual assistant built using Rasa Open Source.

Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X

Share
your bot

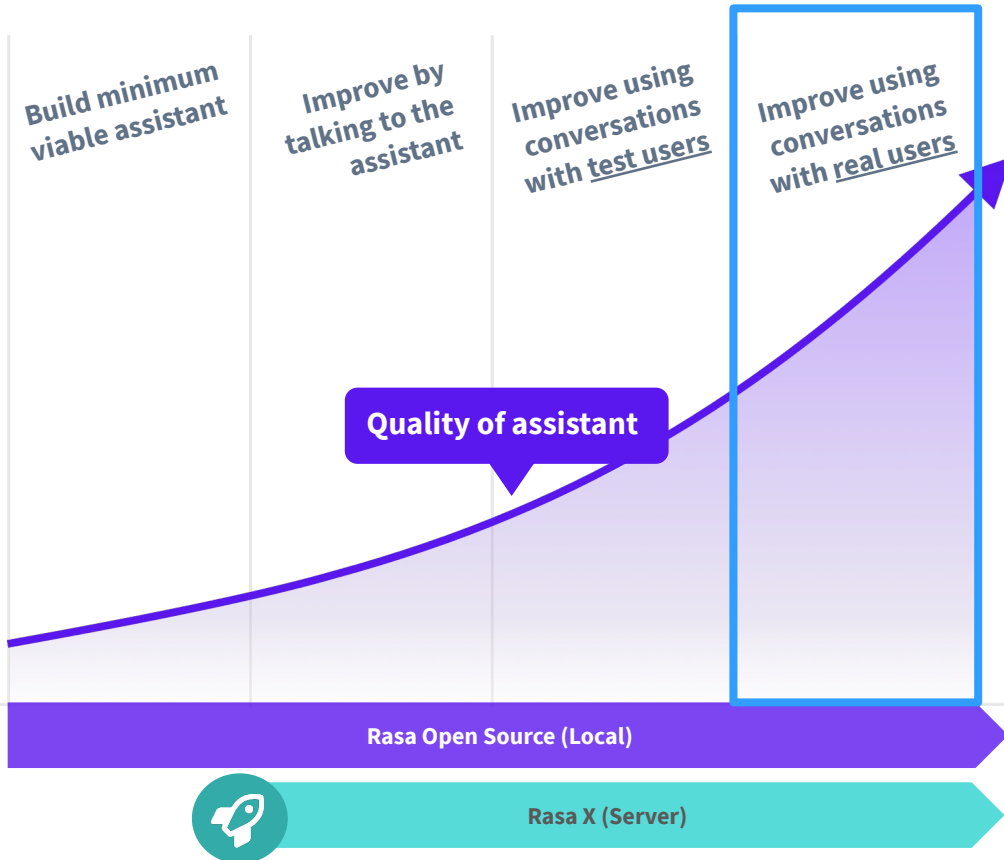


Rasa Open Source is an open source framework for natural language understanding, dialogue management, and integrations.

Rasa X is a toolset used to improve a contextual assistant built using Rasa Open Source.

Build a minimum viable assistant with Rasa Open Source + improve it using Rasa X

Real users
on Telegram



Rasa Open Source is an open source framework for natural language understanding, dialogue management, and integrations.

Rasa X is a toolset used to improve a contextual assistant built using Rasa Open Source.

What's Next?

NEXT STEPS

Get people to chat with your assistant and keep improving it!



Where to go from here?

1. Look at conversations and ask: how does your assistant struggle?
2. Learn how to solve problem from [Rasa Docs](#) and [Community Forum](#)
3. Improve assistant
4. Test and deploy your update
5. Repeat 🤪

Looking forward to tomorrow

Day 1: Setting things up

Day 2: Deep dive into NLU and dialogue management with Rasa Open Source

Day 3: Adding custom actions and implementing forms

Day 4: Deploying and improving your assistant using Rasa X

Day 5: Recap and certification

Presubmit your questions for the Q&A panel! <https://forms.gle/RNbpr6Uv9q7B5Nxi9>

Questions