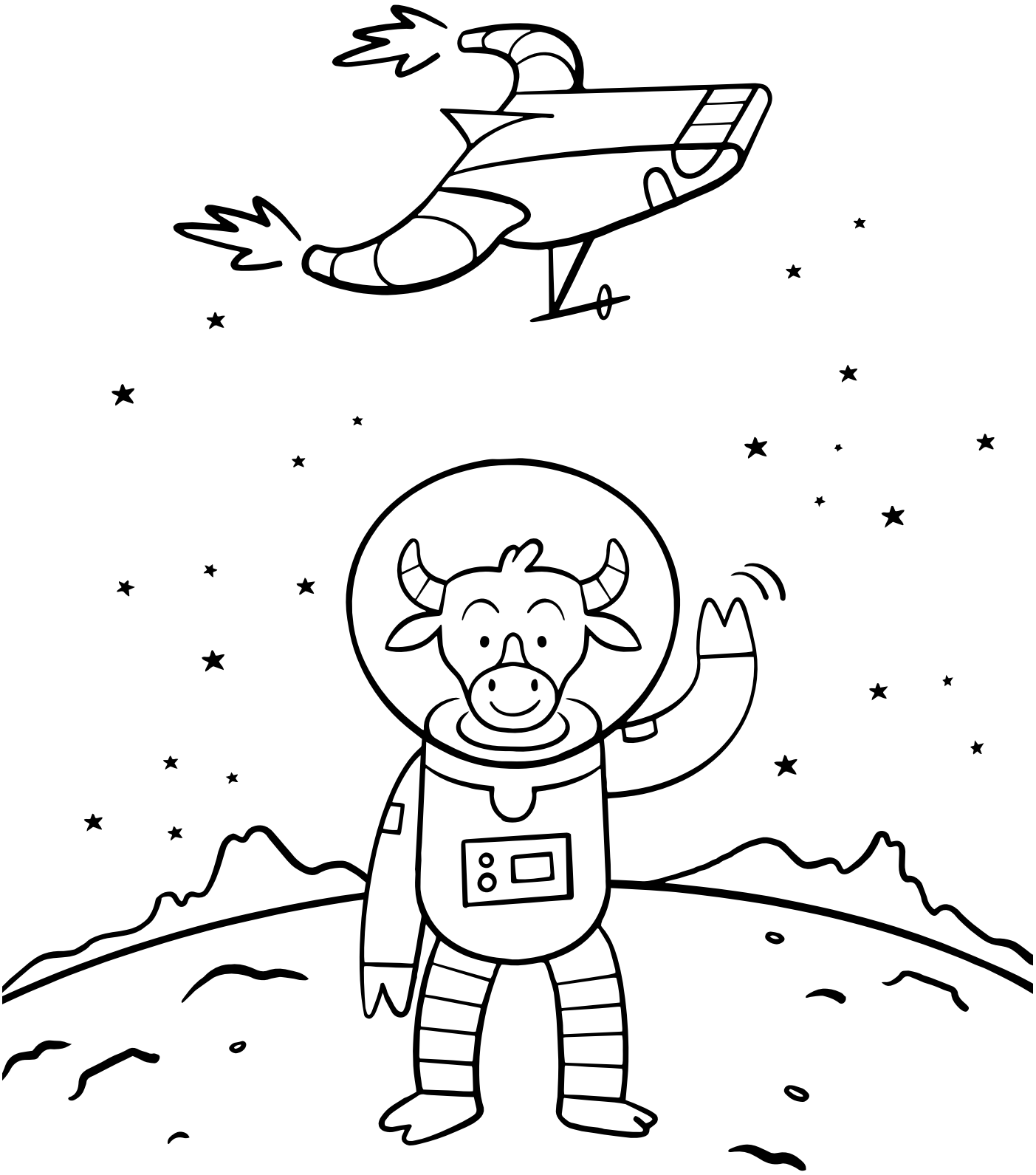
Supported by Red Hat

Ansible

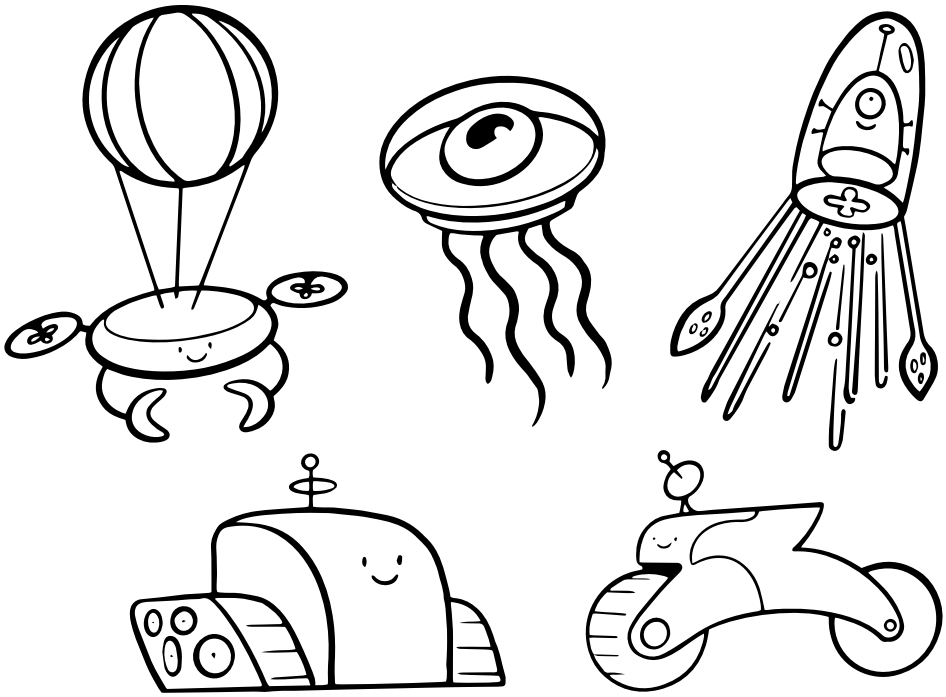
AnsiBull's Galactic Adventures

Script by Máirín Duffy | Illustration by Wildfire | Concept and storyboard by Angela Pagan and Mary Shakshober

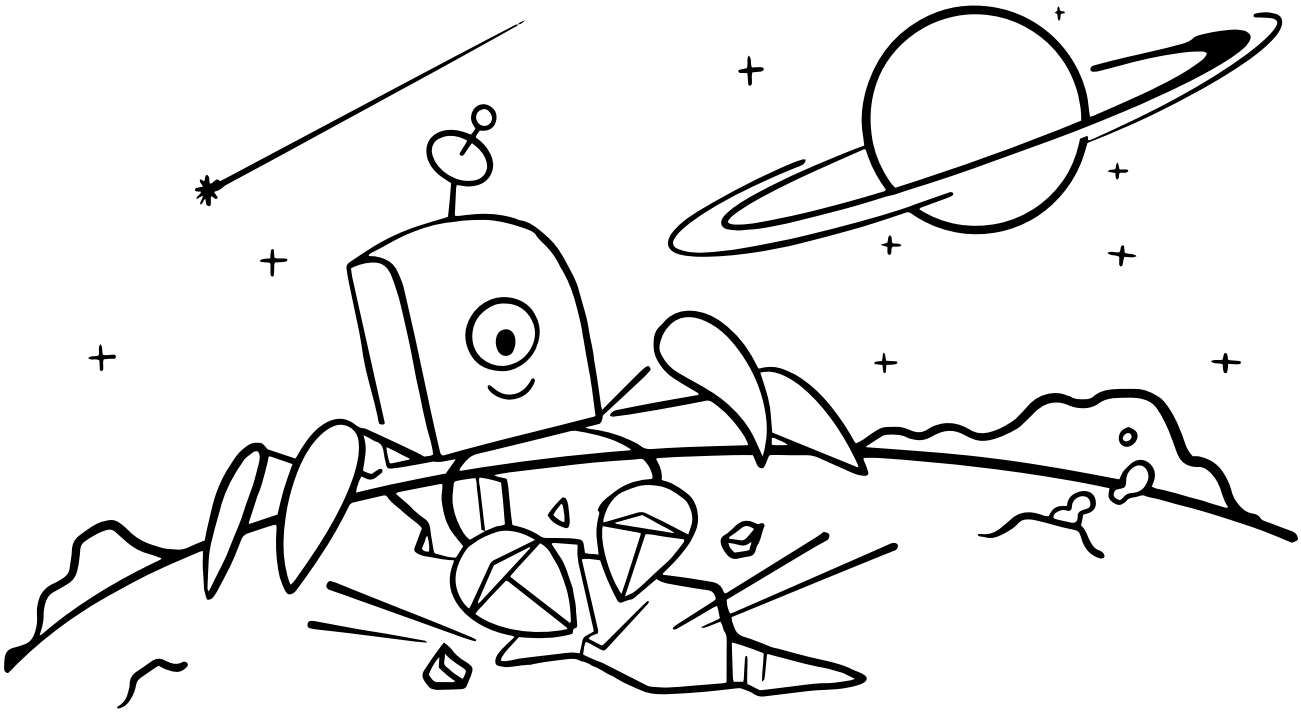


Meet AnsiBull.
He is an astronaut commander and scientist.

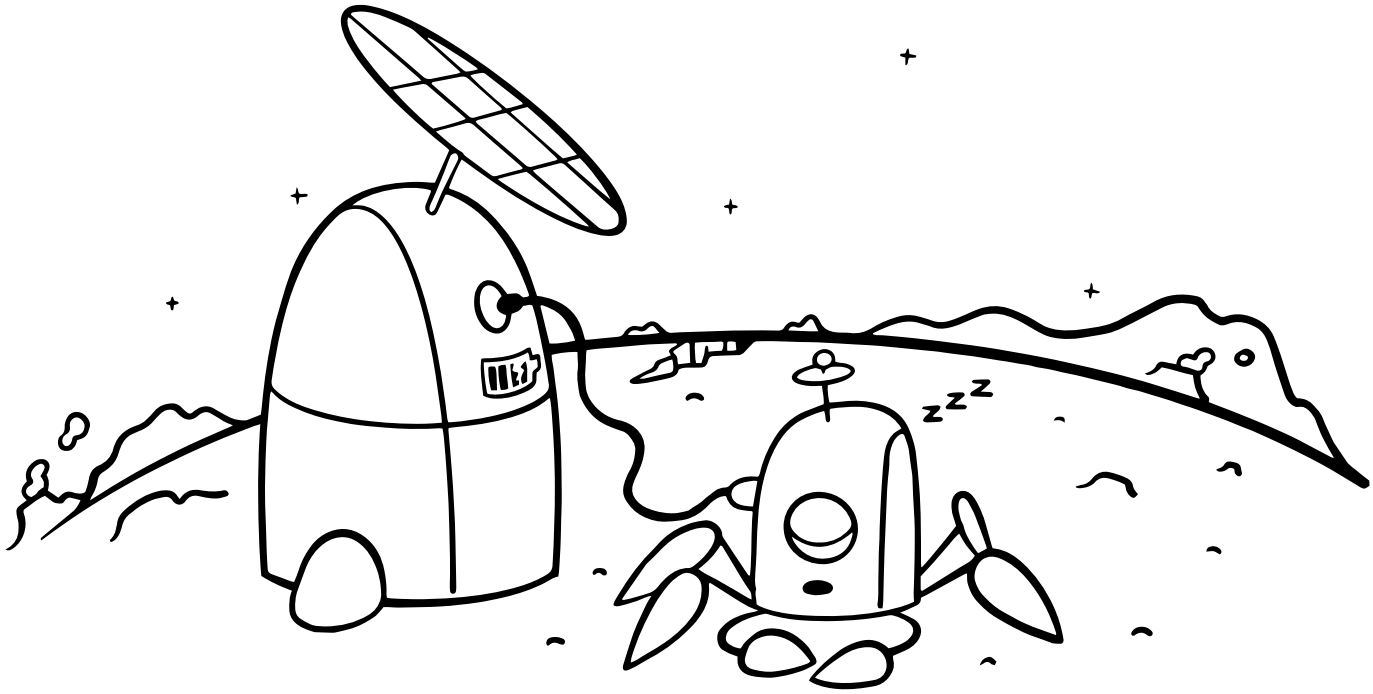
Using Ansible from a control machine aboard his rocket, he manages a vast interstellar network with tens of thousands of surface rovers that collect data for his mission to further our understanding of the universe.



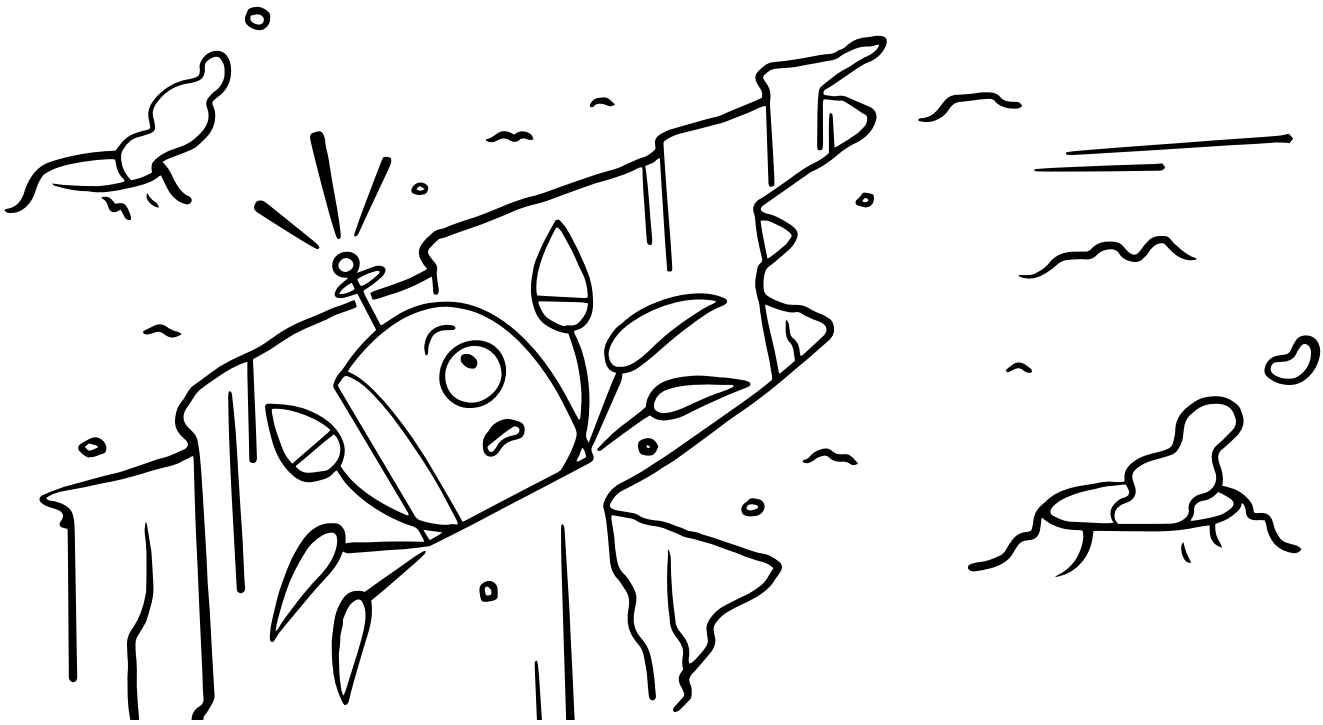
Some rovers are identical, or very similar, in design. Others are customized for the environments they survey. All must be kept updated with the latest data collection commands, configuration, software updates, and security updates that apply to their specific design.



AnsiBull's current project is provisioning a new set of ice-digging rovers to the moons of the gas giants in Earth's solar system.

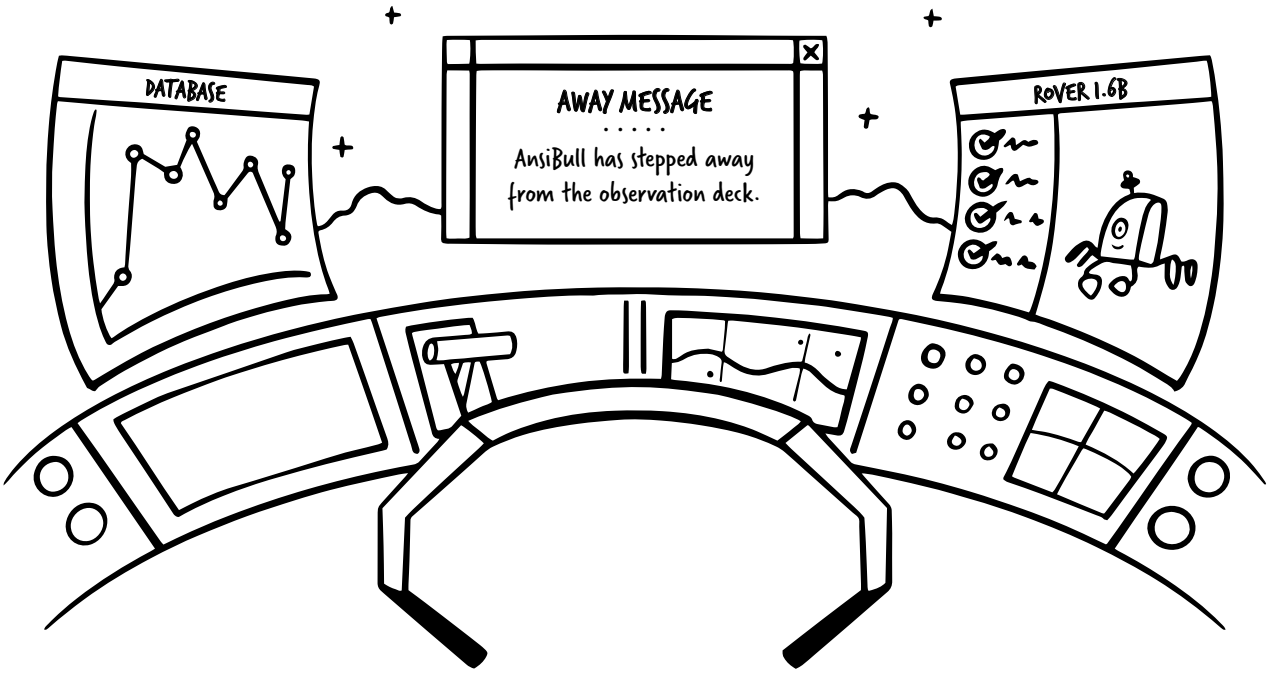


The rovers are remote and may go many years without physical intervention, so stability and resource efficiency is critical.

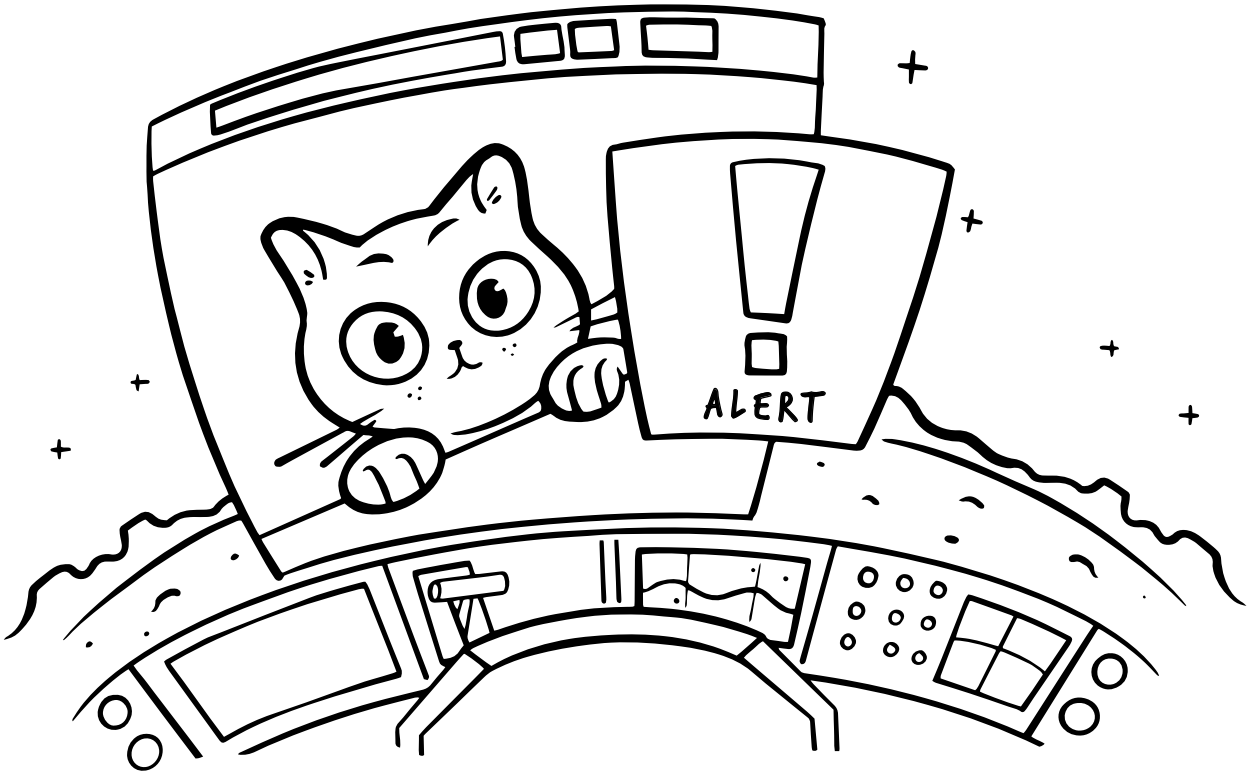


Rather than using an add-on agent to communicate with rovers, AnsiBull uses SSH. Since SSH is a secure communication protocol built in to each rover's operating system, no additional resources are required as they would be with an add-on agent.

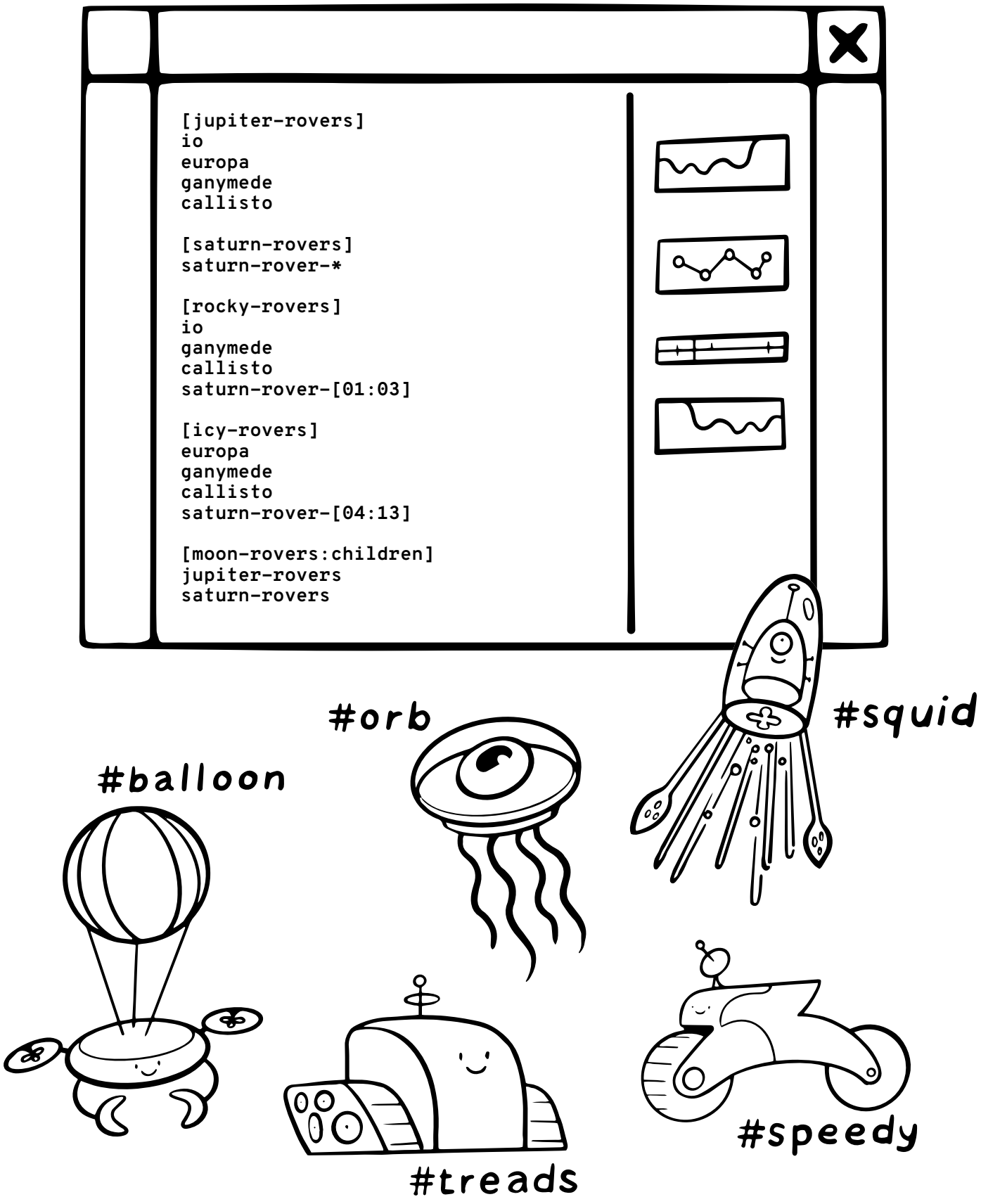
All rovers have SSH built in natively, so no extra assembly is required for reach rover's deployment. SSH is ubiquitous and is used across many computer systems. SSH uses strong encryption for secure communication between the control machine and individual rovers, even across insecure interstellar networks. Thanks to SSH, those mischievous Betelgeusians from Orion's Belt can't replace rover imaging data with furless Rigellian felinoid memes.



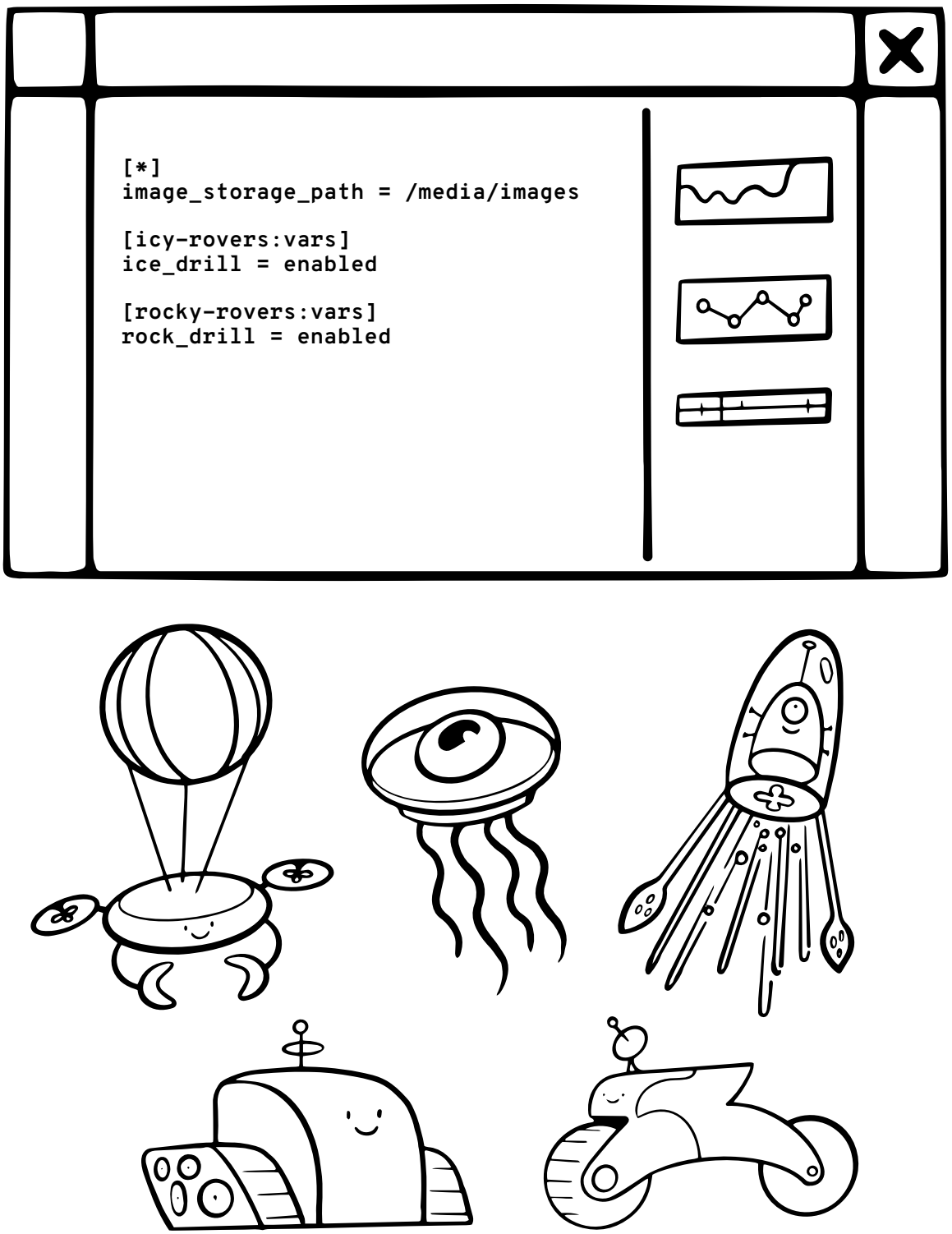
AnsiBull keeps track of thousands of rovers under his watch in an Ansible inventory.



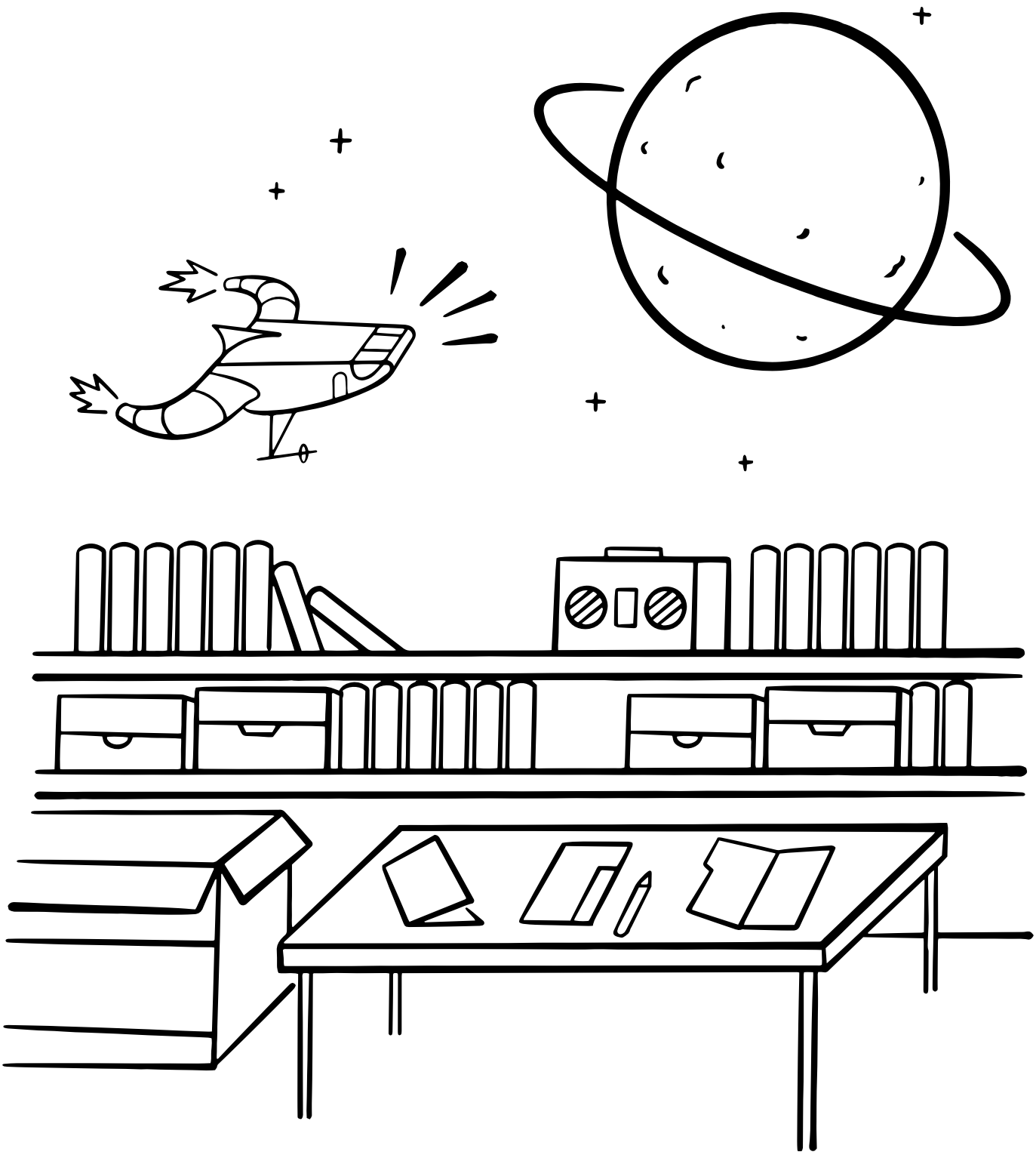
AnsiBull has some of his rovers listed in a static inventory file. He creates his list using rover hostnames or IP addresses, and he lists his rovers logically into groups.



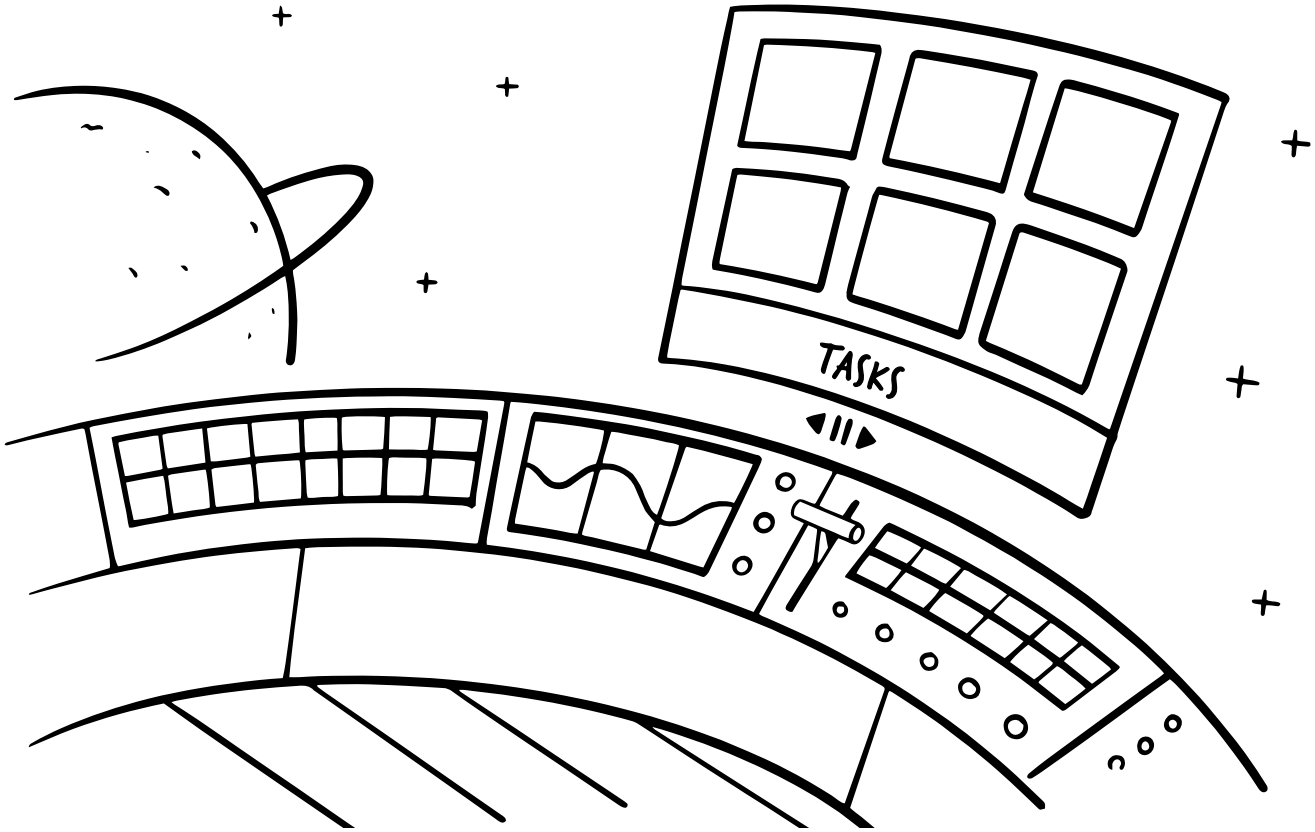
He stores some variables that apply to certain groups in his inventory file, as well.



He can also have a dynamic inventory that updates automatically based on tags applied to systems in external places, like a cloud provider or LDAP.

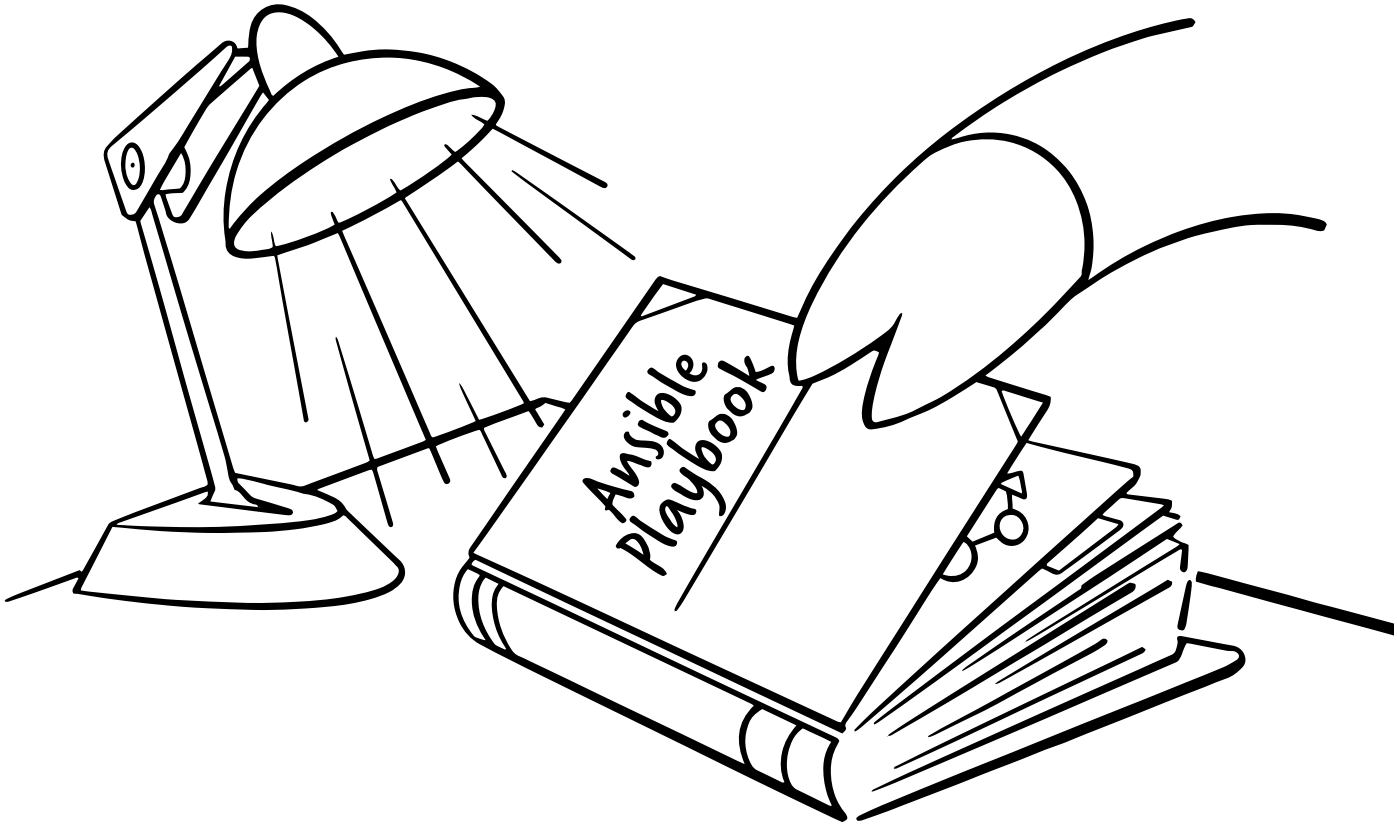


Now AnsiBull has his rovers neatly inventoried and organized. What does he do with them? How does he run commands, or modify configuration files, or install software on them? Modules!



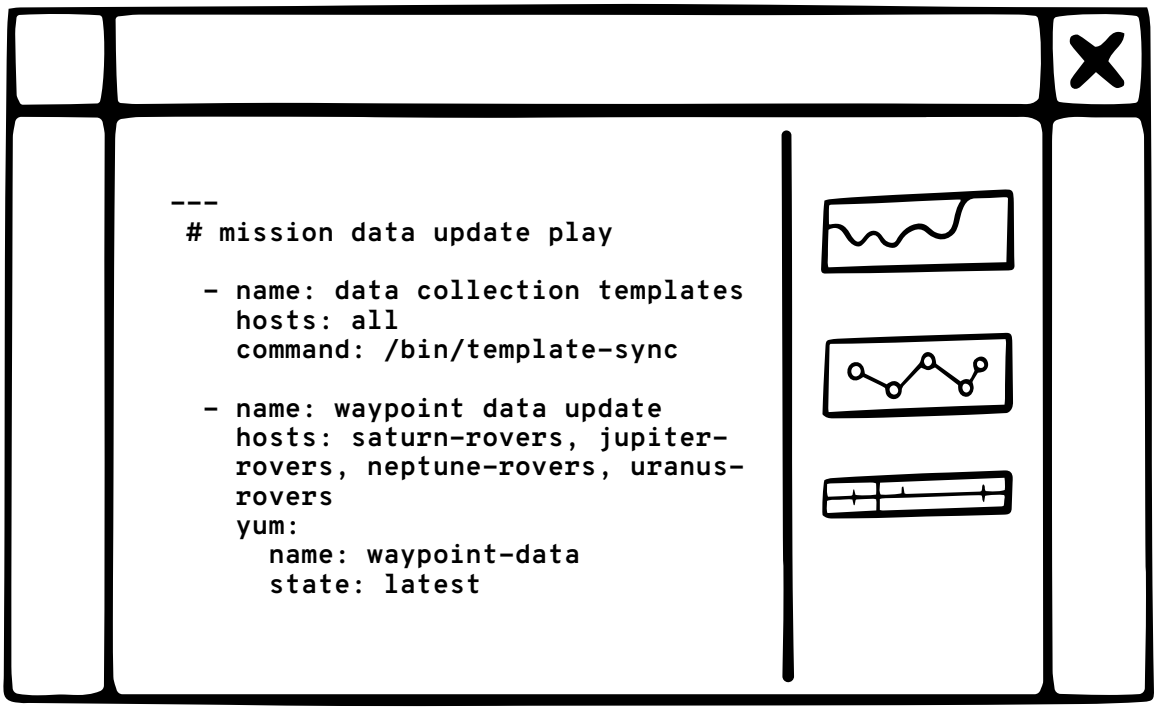
Modules are Ansible plugins that execute tasks. For example: The ping module is a simple diagnostic test to make sure a given rover can be contacted. If successful, a “pong” is returned.

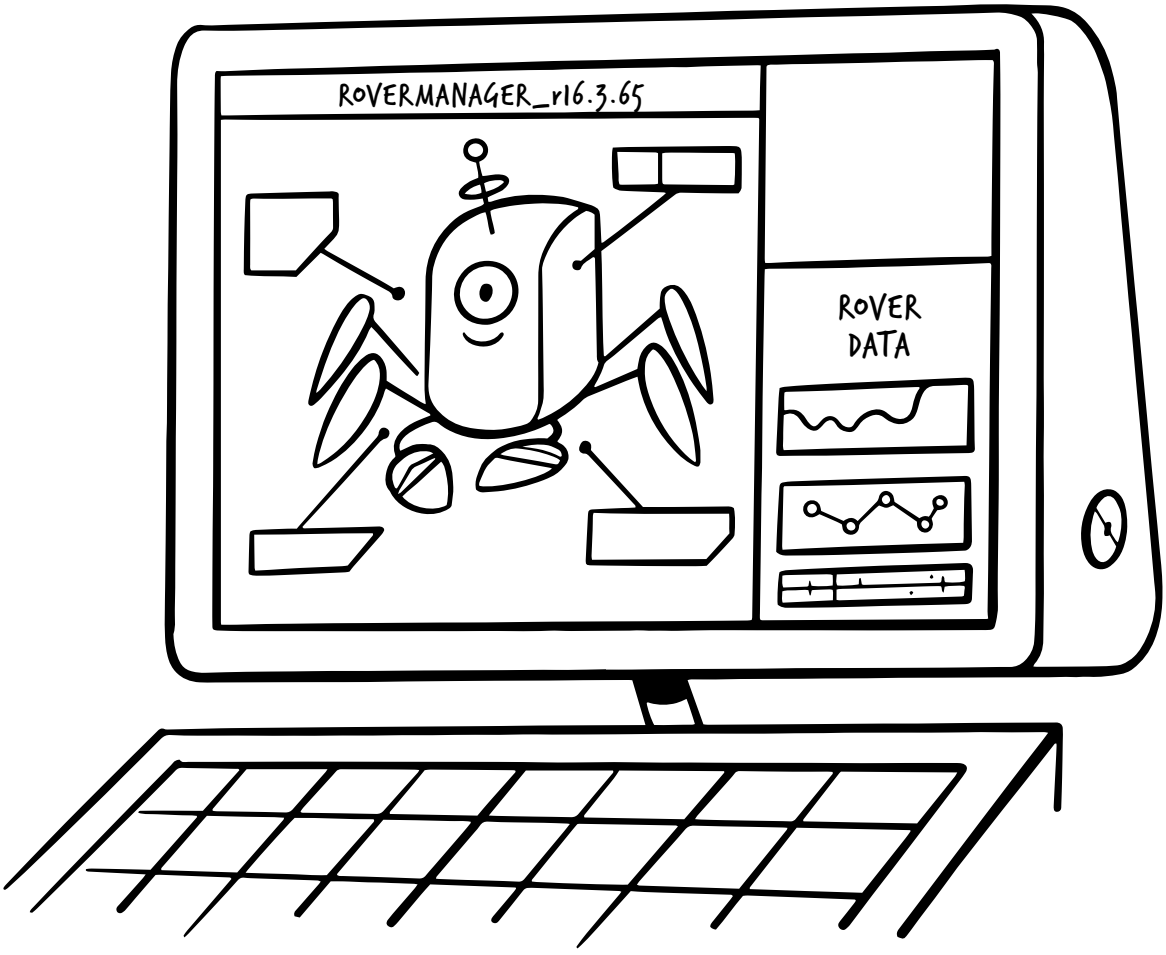
There are hundreds of modules available for Ansible! AnsiBull can even write his own. But how does he use them? Where do they run?



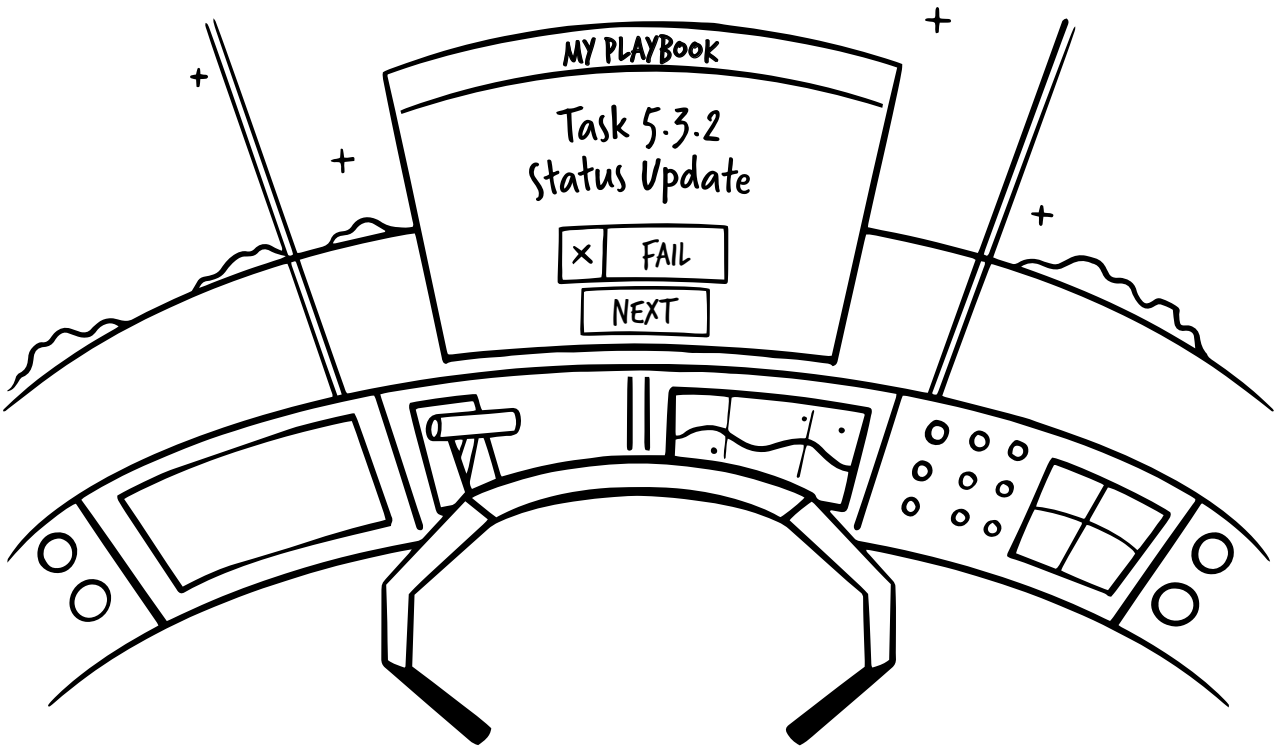
How does AnsiBull consistently manage complex multi-step configuration? How does he setup thousands of rovers so quickly, and up to spec? Ansible playbooks!

A play is a set of tasks. A task is a set of actions, like updating a mission waypoint data package. These are completed with modules, like the yum module for installing packages. Plays are grouped into Ansible playbooks.



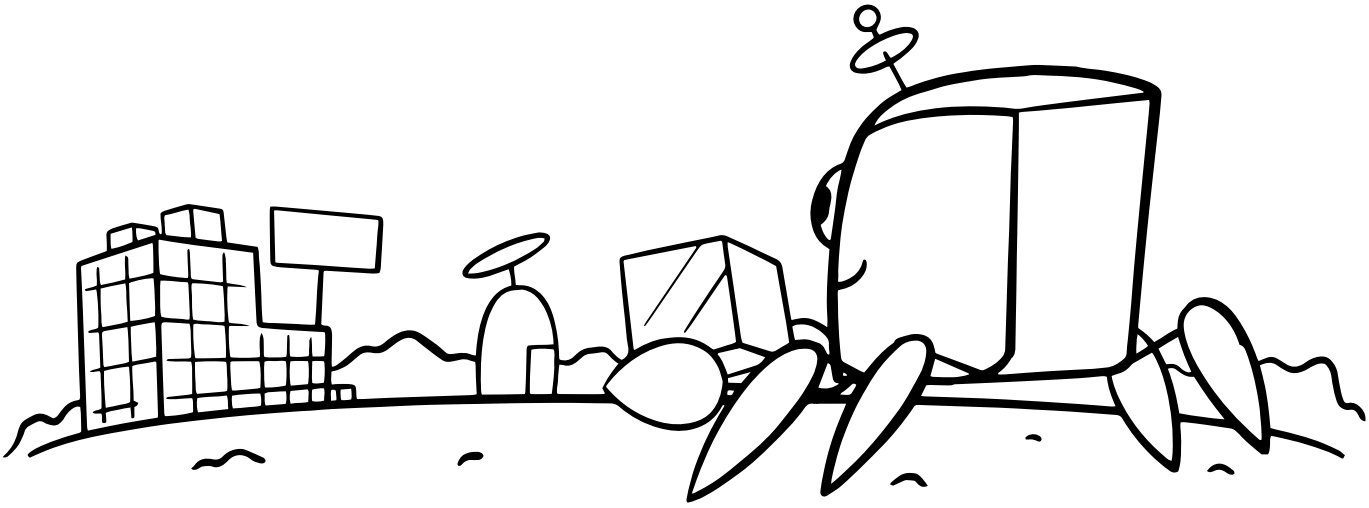


Traditionally, system administrators like AnsiBull would manage a set of systems, trying to script everything the system should do. Ansible has a focus on system state. It does the bare minimum to get the system to the state specified by the playbook.



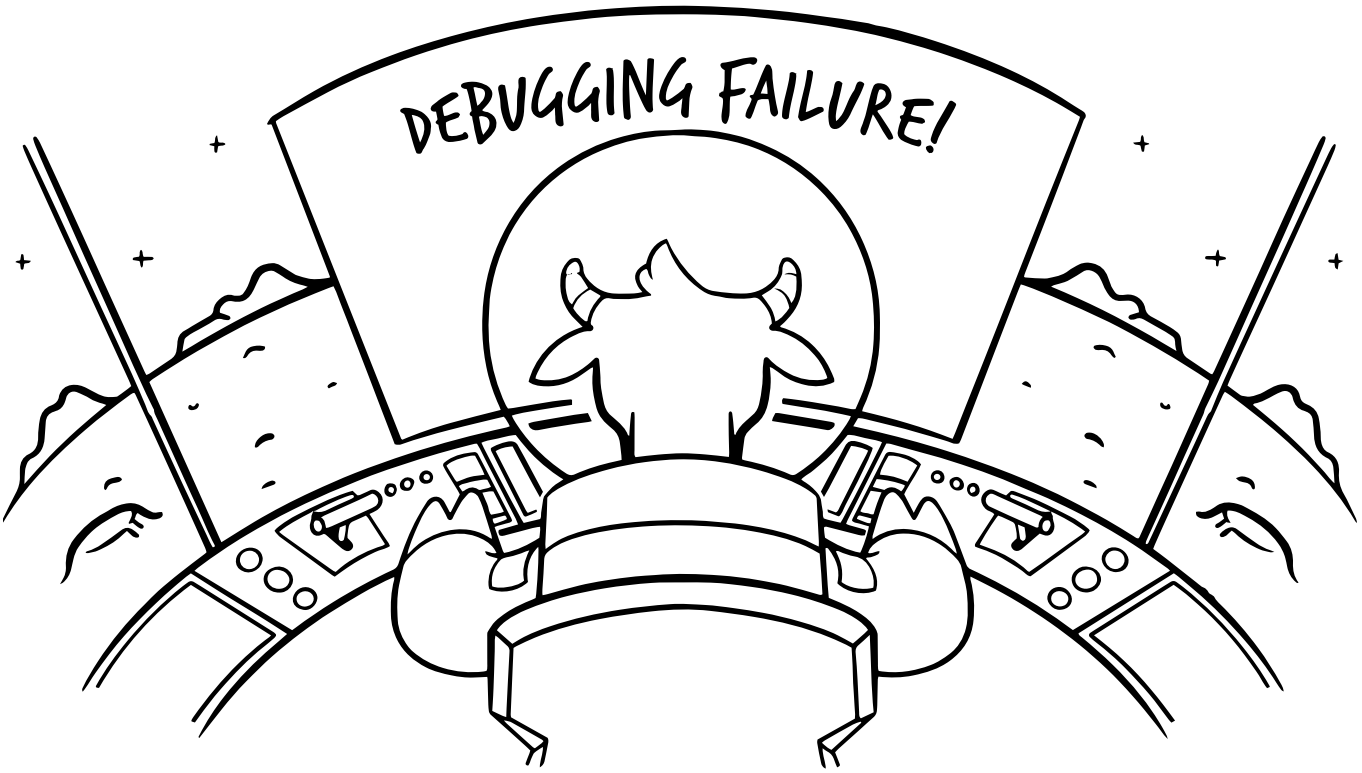
If a task in a playbook fails to run on a particular system, it's exempt from the remaining tasks in the playbook on that run.

AnsiBull then receives a status notification that saturn-rover-05 didn't complete the playbook. He runs the playbook again so that any systems that happened to fall in a ditch or get attacked by a Rigellian felinoid catch up on anything they missed. Rovers that already successfully completed all the tasks in the playbook's plays aren't affected.

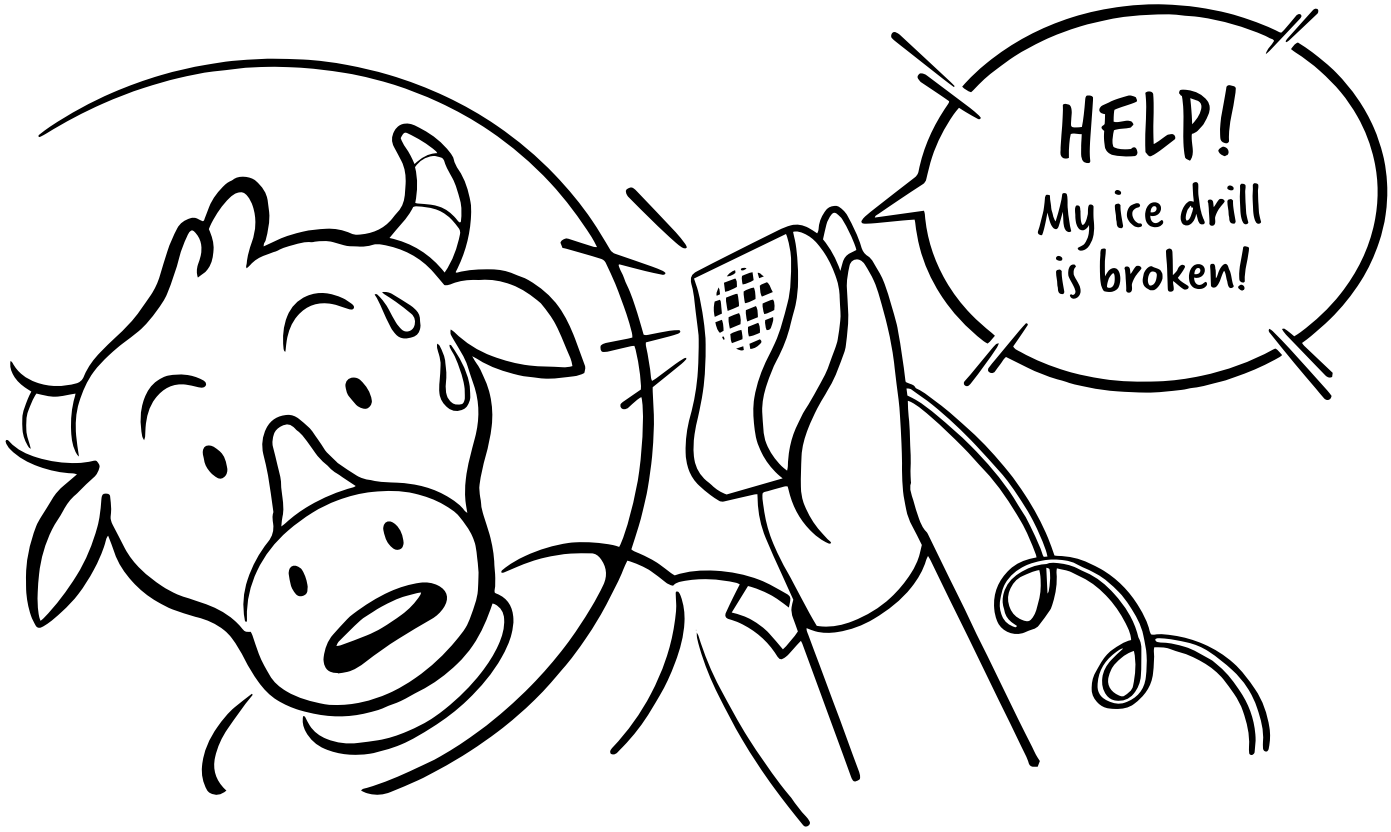


Ansible is a different way of approaching system management. It focuses on the state that a system must be in; it provides an ice plant rover, a fully-updated rover, and so on. Actions needing to be run to get rovers to those states are handled at a lower level, in tasks and plays.

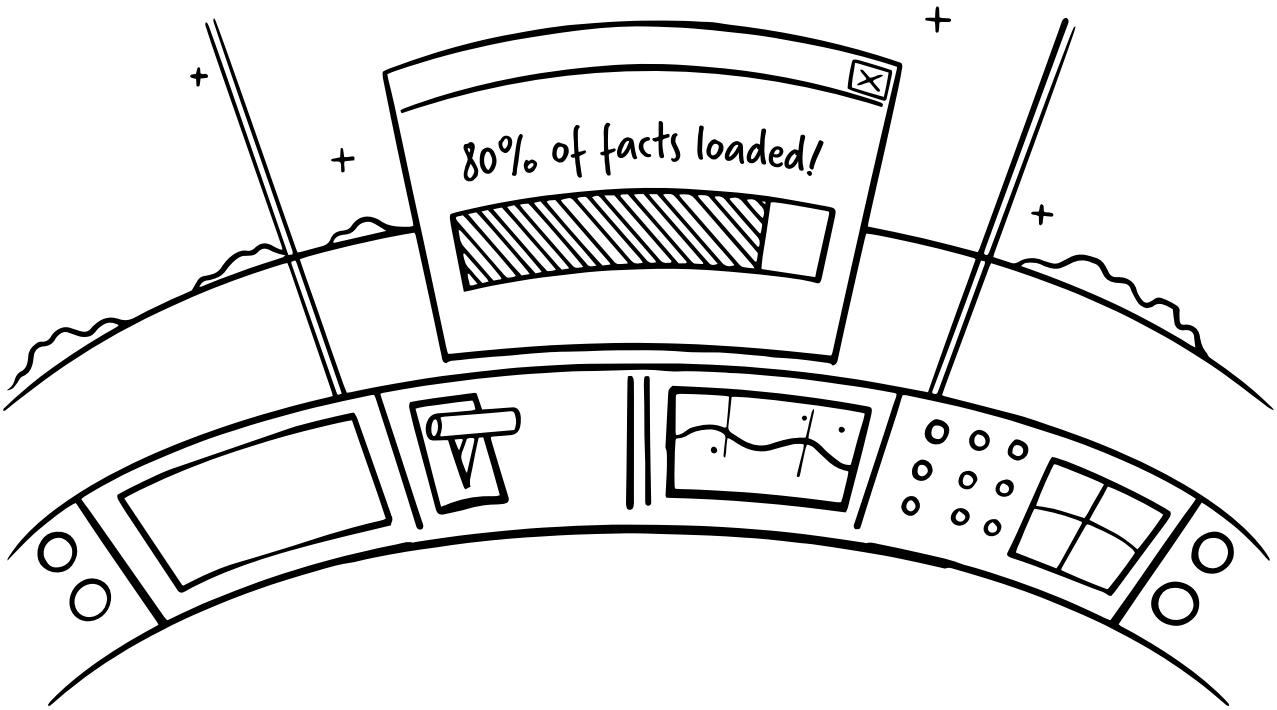
AnsiBull tries to run the playbook on the rover again, but receives a notification that an error occurred. The ice drill on the rover can't be enabled... it was damaged in the cliff fall. Oops.



AnsiBull uses Ansible's interactive command interface to debug the ice drill remotely, but can't figure it out.

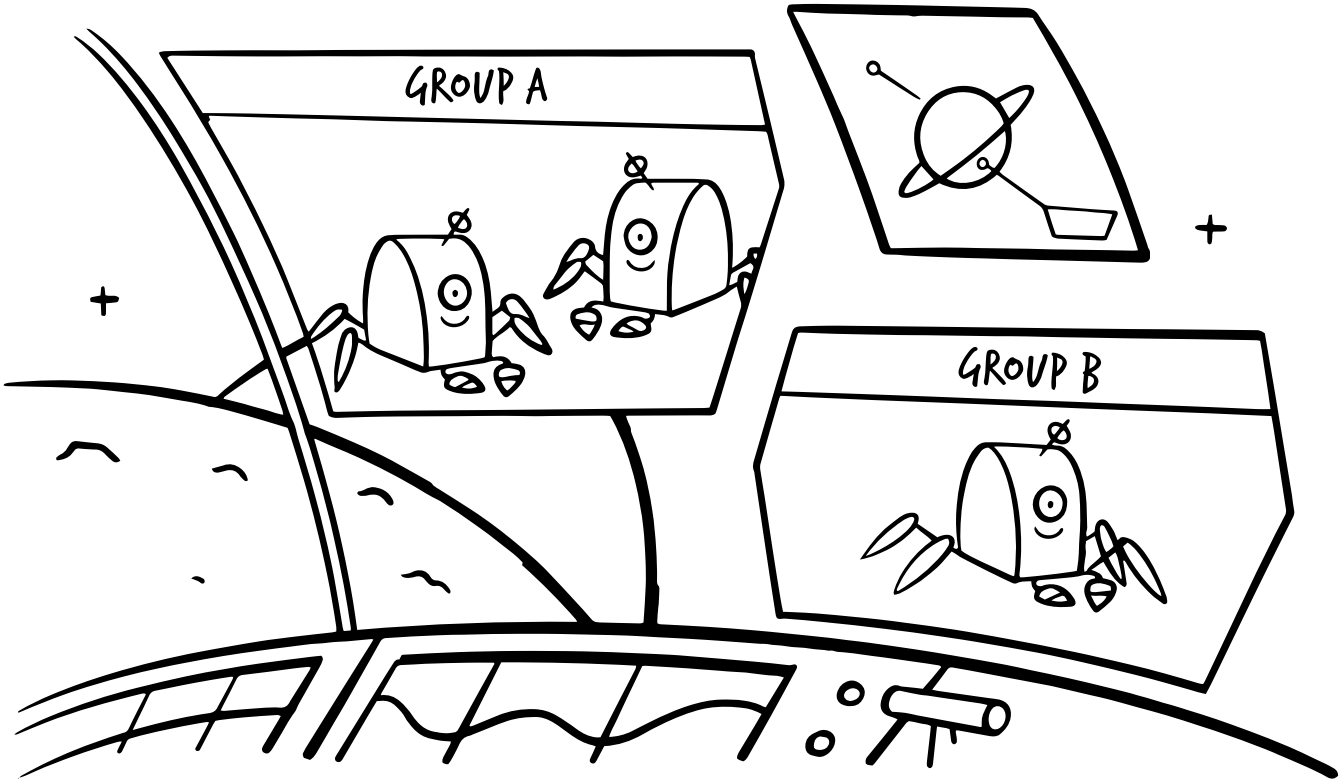


Time to call ice drill customer support for help! Uh-oh. Is the drill still under warranty? Oh, and what brand drill is it? Who should AnsiBull call?

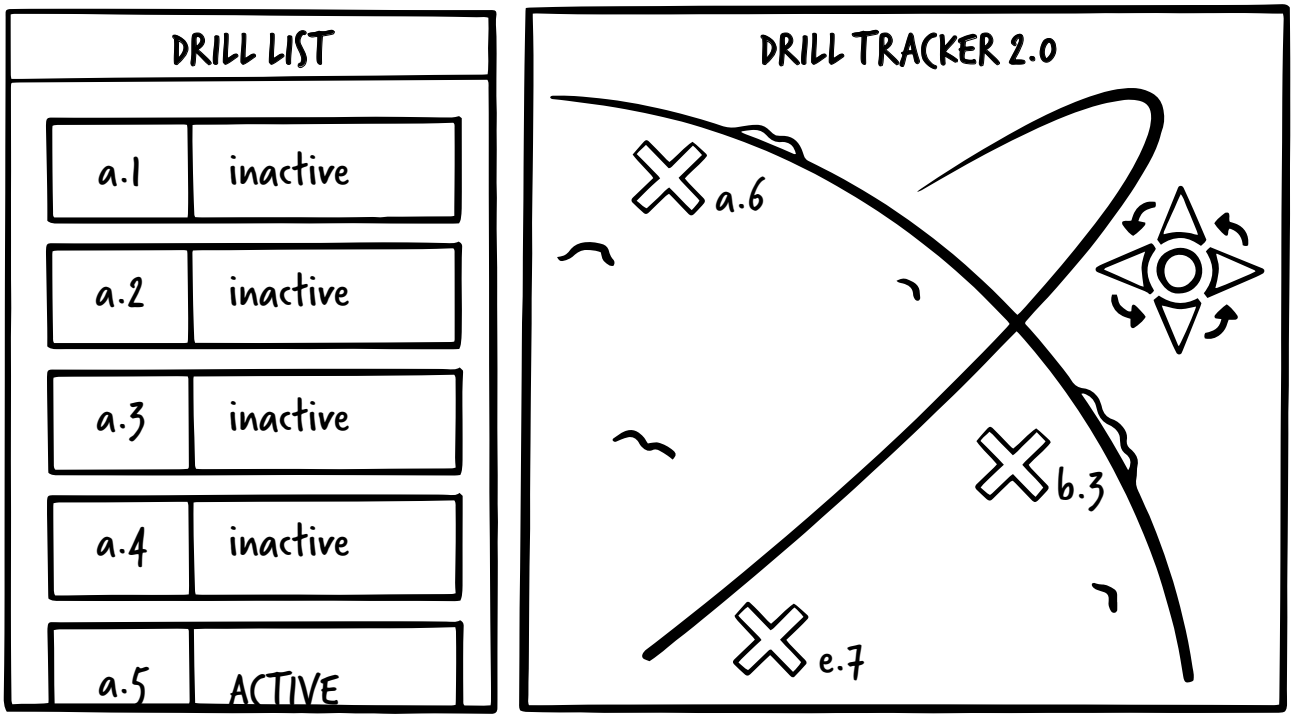


Ansible has a feature that gathers facts about systems. Some facts are gathered automatically based on information provided by the system. Some facts are custom values created by scripts.

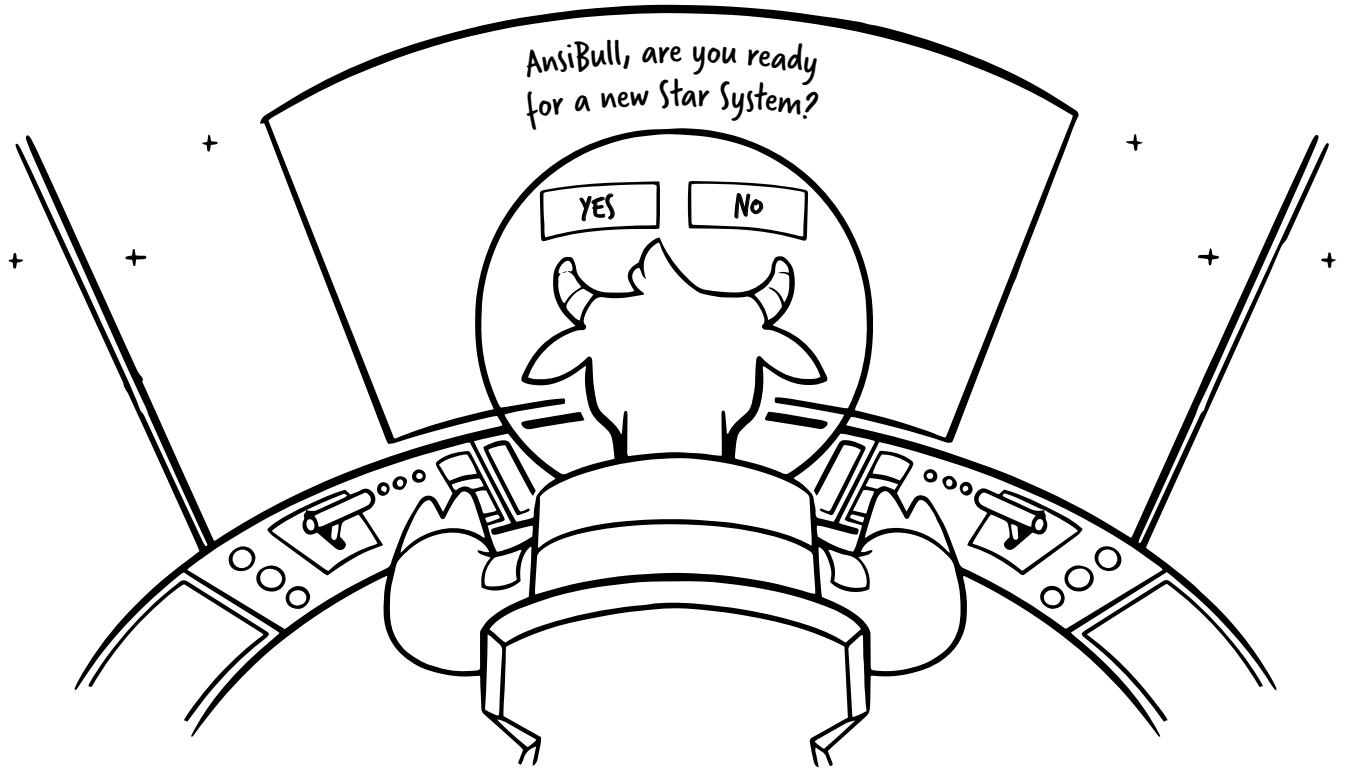
Using the facts system, AnsiBull finds the right hardware vendor, the serial number for the drill, and the warranty lapse date. It's still under warranty! He calls the vendor and finds out there's been a recall on the drill.



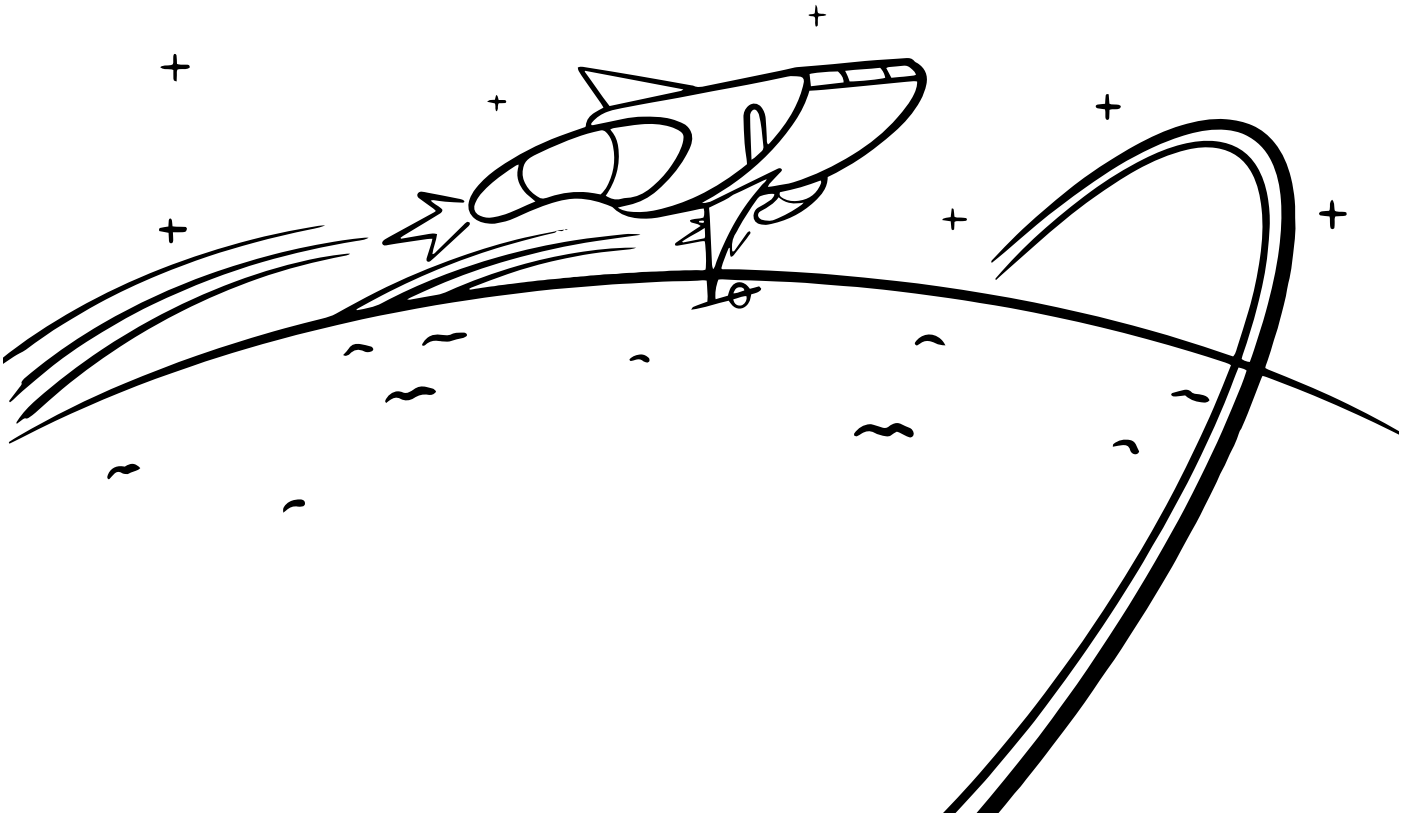
Which rovers have this kind of ice drill? Using Ansible facts, he creates a group of all rovers with that drill model, gets the list of serial numbers, and orders replacement drills from customer support.



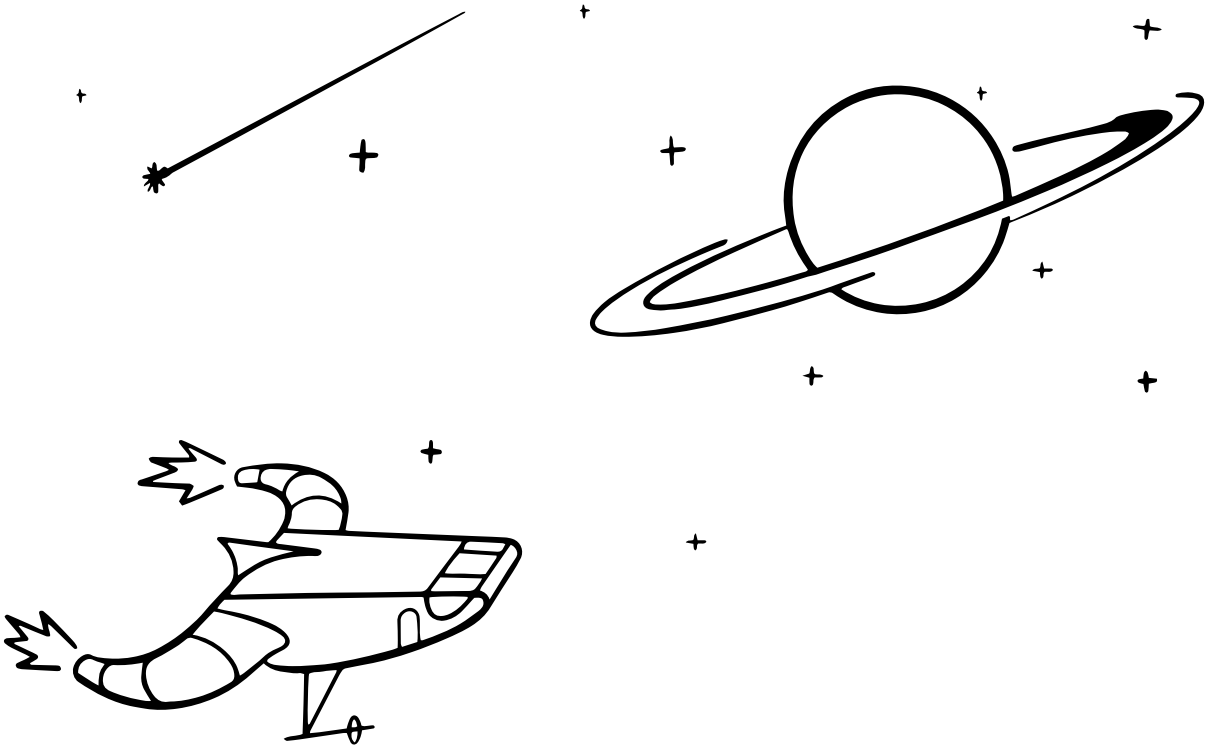
He also generates a list of the locations of all recalled drills so he can plan out his trips to install replacements.



Thanks to Ansible, a drill disaster has turned into an easy fix. The AnsiBull has finished his ice-digging rover deployments, and is ready to move on to other star systems.



To learn more about Ansible, please visit http://ansible.com



.....

For more open source activities and inspiration,
visit redhat.com/colab.