Getscreen.me

Self-Hosted Server Guide

Self-Hosted Server is a packaged edition of the Getscreen.me remote access platform for deployment in a customer's own infrastructure. This guide describes the system, its installation, and further maintenance.

November 16, 2026

Table of contents

1. About the Self-Hosted Server 5
2. Self-Hosted Server Stack 6
2.1 Main Server 6
2.1.1 Download Component 7
2.1.2 Proxy Component 7
2.1.3 Image Component 7
2.1.4 Storage Component 7
3. System Requirements 8
3.1 Hardware Resources 8
3.2 Network Resources 8
3.3 Peripheral Tools 8
3.4 Additional Proxy Components 8
4. Self-Hosted Server Installation 10
4.1 Run Install Code 10
4.2 Issue SSL-certificate 10
4.3 Setup SMTP Server 11
4.4 Launch System 11
4.5 Check Errors 11
4.6 Login to Server 12
5. Self-Hosted Server Maintenance 13
5.1 Backup Data 13
5.2 Update System 13
5.3 Reset Admin Password 13
5.4 Video Recordings Storage 14
6. Main Server Configuration 15
6.1 Listening Ports 15
6.2 Database 15
6.3 Inter-Component Communication 15
6.3.1 Authorization Token 16
6.3.2 Component Credentials 16
6.3.2.1 Download Component 16
6.3.2.2 Proxy Component 16
6.3.2.2.1 STUN Servers 16
6.3.2.2.2 TURN Servers 17
6.3.2.2.3 WebSocket Servers 17

6.3.2.3 Image Component 17
6.3.2.4 Storage Component 18
6.4 Communication Tools 18
6.4.1 Mail Server 18
6.4.2 Telegram Bot 19
6.4.3 Automatic Text Translation 19
6.4.3.1 Google Translate 19
6.4.3.2 Azure Translator 19
6.5 Business Settings 20
6.5.1 Video Capture Settings 20
6.5.2 Notification Settings 20
6.5.3 Features Limits 21
6.6 Security Settings 21
6.6.1 Captcha 21
6.6.2 Unregistered Access 21
6.6.3 Password Settings 22
6.6.4 Old Browsers 22
6.6.5 HSTS (HTTP Strict Transport Security) 23
6.6.6 CSP (Content Security Policy) 23
6.7 Network Settings 23
6.7.1 Reverse Proxy 23
6.7.2 Log HTTP Requests 24
6.7.3 Dashboard App Ports 24
7. Adding a Proxy Component 25
7.1 System Requirements 25
7.2 Installation 25
7.2.1 Prepare Environment 25
7.2.2 Prepare Config 25
7.2.3 Run Container 26
7.3 Config Main Server 26
8. Main Server TLS Encryption 27
8.1 Minimum Protocol Version 27
8.2 Maximum Protocol Version 27
8.3 Ciphers 27
9. Custom Domain Setup 29
9.1 Endpoint Types 29
9.2 Domain Setup 29
9.2.1 Main Server Configuration 29

9.2.2 Adding DNS Records 30
9.2.3 License Update 30
9.3 Before You Start 30
9.3.1 SSL Certificate Issuance 30
9.3.2 Reinstalling Agent Applications 30
9.3.3 Component Dependencies 30
9.3.4 Single Sign-On (SSO) 30
9.3.5 Importing Technicians from Active Directory 30
10. Telegram Bot Setup 31
10.1 Create Bot 31
10.2 Configure Bot 32
10.2.1 Domain name 33
10.2.2 Webhooks 33
10.3 Configure Main Server 34
10.4 Main Server Restart 34
10.5 Check 34

1. About the Self-Hosted Server

Self-Hosted Server is a specially packaged version of our cloud-based remote access platform, adapted for deployment within a customer's own infrastructure and distributed as a separate product edition.

The key advantages compared to the cloud version are:

- **Autonomous Operation** — the solution can operate within a fully isolated local network perimeter, even without access to the public Internet.
- **Secure Environment** — no one except the customer has access to the connection management server and storage containing sensitive data.
- **Connection Speed and Quality** — the customer can independently manage network performance and conditions by configuring the network and deploying proxy servers in required locations.
- **Full Control** — the customer is fully responsible for maintaining and controlling both the software and hardware environments, without external dependencies or third-party risks.

In most cases, the Self-Hosted solution is designed for enterprise-level customers, as it requires sufficient IT expertise for deployment and ongoing maintenance.

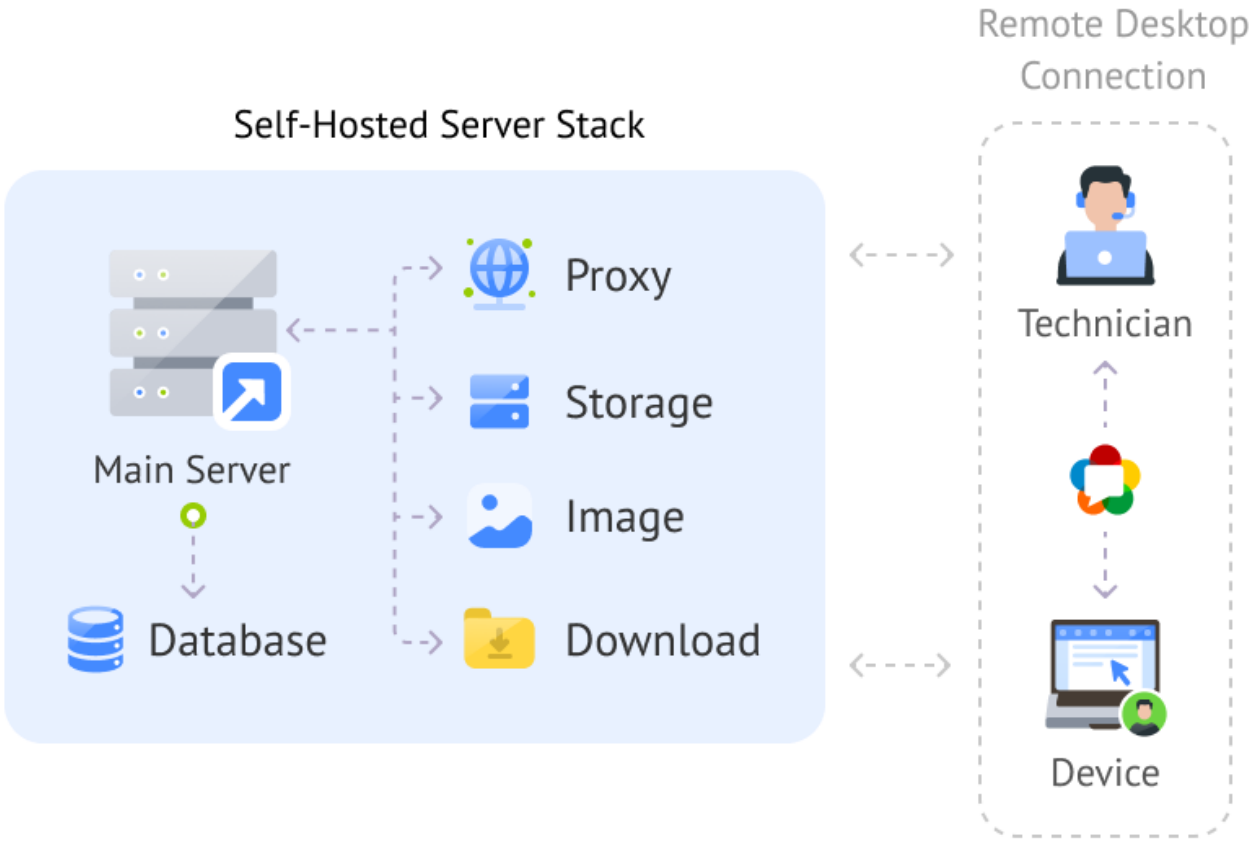
This guide provides reference information about the contents of this edition (hereinafter referred to as the **System**), its installation, and further maintenance. All materials assume that you have adequate experience working with the Linux operating system and are familiar with containerization technologies.

- **Server Stack** — Learn general information about the system components and their roles.
- **System Requirements** — Learn about the environment system requirements required for optimal system performance.
- **Installation** — Review the detailed installation and deployment guide to get started.
- **Maintenance** — Learn about technical maintenance activities required to ensure uninterrupted system operation.
- **Advanced Settings** — Learn how to configure all system features to organize efficient system operation.
- **Change Log** — Stay up to date with new Self-Hosted Server releases in the main changelog section.

2. Self-Hosted Server Stack

The Self-Hosted Server is a set of server components, each responsible for a specific function required to organize and manage remote access between technicians and devices. In addition to these components, the deployment includes a PostgreSQL database (version 14) used for data storage.

During operation, all system components interact with the main server and/or the database. The overall system architecture can be represented by the following diagram.



2.1 Main Server

This is the central brain of the system. It manages all other components and performs the most critical function: connecting the technician's browser with the agent application running on the remote device.

The main server is a web server and serves as the entry point to the system. It is bound to the domain and ports used by both the technician's browser and the agent application.

Other key responsibilities of the main server include:

- Maintaining a persistent service connection with remote devices
- Providing access to the technician's personal dashboard
- Delivering notifications, system emails, and messages
- Managing SSL certificates
- Logging all system activity
- Ensuring secure access

- Registering and managing the entire system inventory, including:
  - Technician accounts
  - Devices and device groups
  - Access control and permission system
  - Quick support sessions

2.1.1 Download Component

A subordinate component responsible for managing, storing, and delivering agent application distributions.

Its functions include:

- Storing all versions of agent application installers
- Providing installers upon request from the main server
- Generating one-time installers for invitation-based access
- Generating branded versions of installers
- Updating existing installers

2.1.2 Proxy Component

A subordinate component used to relay the video stream from a remote device to the technician when a direct P2P connection cannot be established due to network restrictions on either side.

This component can be scaled horizontally to improve connection speed and enhance the video quality of remote sessions.

2.1.3 Image Component

A subordinate component responsible for handling desktop preview images from registered devices.

Its functions include:

- Receiving images from the agent application
- Providing images to the main server for display in the technician's dashboard
- Storing and managing image data

2.1.4 Storage Component

A subordinate component responsible for handling session video recordings.

Its functions include:

- Receiving video recording data from the technician's browser
- Encoding video data into files
- Providing recorded files upon request by the technician

3. System Requirements

3.1 Hardware Resources

The system requires a virtual machine with the following characteristics:

- Operating system: **Debian/Ubuntu** - Ubuntu 24.04/Debian 12 recommended
- RAM: **8 Gb**
- Disk: **32 Gb**
- CPU: **4 cores**

3.2 Network Resources

The virtual machine must have 1 real **IPv4 address** and **domain name**.

If a firewall is used, the following network ports must be open for incoming connections:

Port	Protocol	Destination
`80`	`TCP`	Redirect to a secure HTTPS connection
`443`	`TCP`	Access to the web interface of the system, signal connection with agents
`3478`	`TCP` and `UDP`	Connections via WebRTC protocol
`40000-60000`	`UDP`	Connections via WebRTC protocol (media data)
`5031`	`TCP`	Connecting to agents via Websocket protocol
`5032`	`TCP`	Connects to the Download Component. Used by Main Server.
`5033`	`TCP`	Connects to the Image Component. Used by agents
`5034`	`TCP`	Connects to the Storage Component. Used by agents and the Main Server.

as well as access to the following resources:

Host	Destination
`https://updater.getscreen.me`	Obtaining and updating a license key, access to agent application distributions
`https://updater.getscreen.me:5000`	Access to containers and archive for installation

3.3 Peripheral Tools

For the system to function fully, you will need a mail server to send service emails. You can deploy your own mail server or use cloud solutions such as MailChimp or others.

3.4 Additional Proxy Components

When scaling the system "wide", you can deploy additional Proxy component. The system requirements for them are as follows:

- Operating system: **Debian/Ubuntu** - Ubuntu 24.04/Debian 12 recommended

- RAM: **2 Gb**
- CPU: **2 cores**
- Drive: **16 Gb**
- **Public IP address** with at **least 100 Mbps** of bandwidth

If a firewall is used, the following network ports must be open for incoming connections:

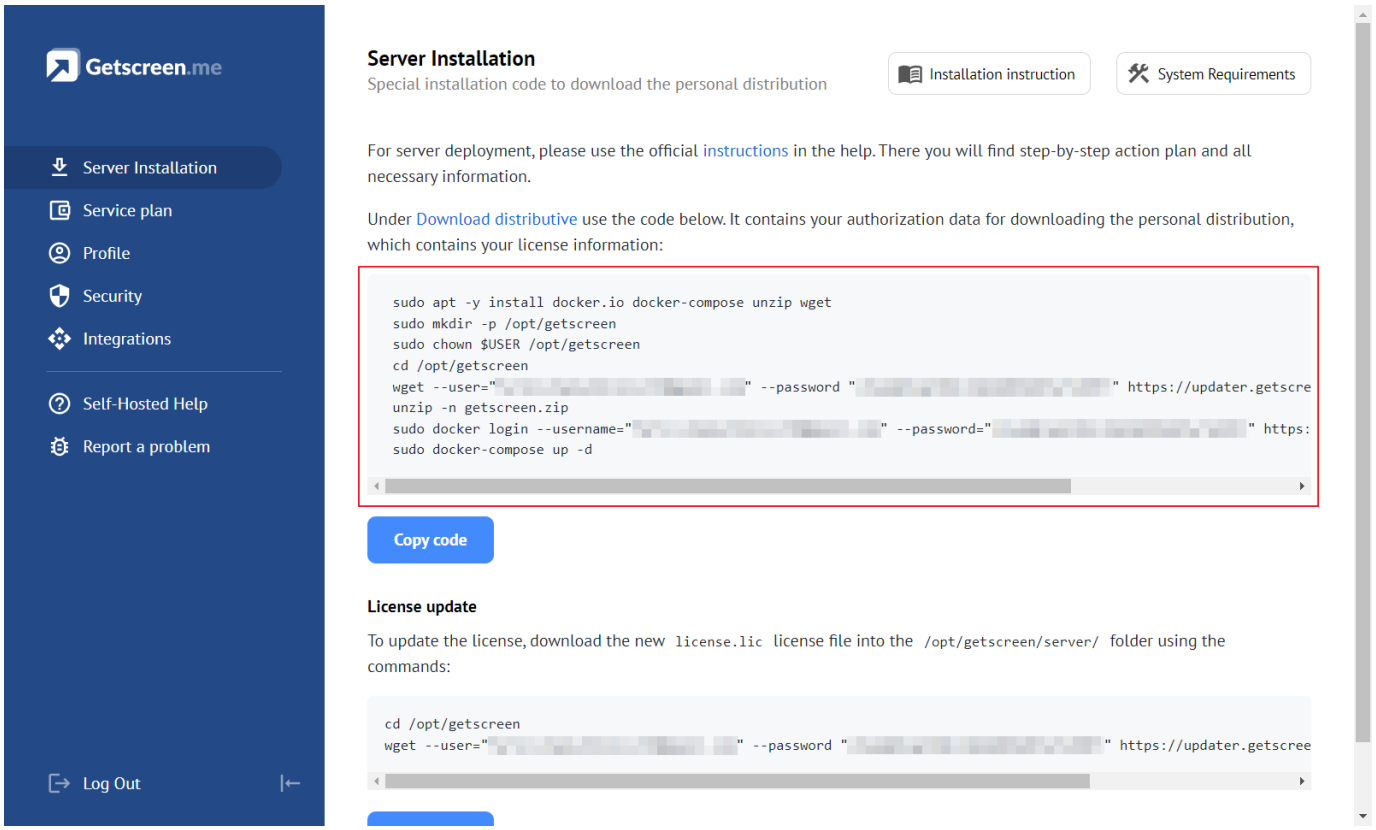
Port	Protocol	Destination
`3478`	`TCP` and `UDP`	Connections via WebRTC protocol
`40000-60000`	`UDP`	Connections via WebRTC protocol (media data)

4. Self-Hosted Server Installation

To install, follow this instruction consisting of several simple steps in order.

4.1 Run Install Code

In the personal account of your account, go to the **Server Installation** section, copy the provided code, and run it in your environment:



The installation code will perform the following steps:

- Installs the minimum required environment packages: `docker`, `unzip`, `wget`.
- Creates the `/opt/getscreen` directory.
- Downloads and unpacks your personal system image bound to your license into it.

4.2 Issue SSL-certificate

A self-signed SSL certificate is included in the standard delivery of the system. It can be found in the following files:

Path	Description
`/opt/getscreen/cert/server.crt`	file in `PEM` format with a certificate (or a chain of certificates)
`/opt/getscreen/cert/server.key`	file in `PEM` format with a private key

To ensure that browsers do not generate warnings about insecure connections, you must obtain a full-fledged certificate. This can be purchased from a certificate authority, or allocated for free from Let's Encrypt .

Run the following commands, replacing `<hostname>` with your domain name. During execution, follow the instructions the system gives you. Access to your domain's DNS server will be required — make sure to take care of this in advance.

```
sudo apt -y install certbot
sudo certbot certonly --standalone -d <hostname>
```

After completing all the actions, the files of the obtained certificate will lie in the following directories:

Path	Description
`/etc/letsencrypt/live/<hostname>/fullchain.pem`	certificate with full chain of certifications
`/etc/letsencrypt/live/<hostname>/privkey.pem`	private key

Copy these files to the `/opt/getscreen/cert/` directory by running the following commands:

```
sudo cp --copy-contents /etc/letsencrypt/live/<hostname>/fullchain.pem /opt/getscreen/cert/server.crt
sudo cp --copy-contents /etc/letsencrypt/live/<hostname>/privkey.pem /opt/getscreen/cert/server.key
```

4.3 Setup SMTP Server

Sending mail is required to deliver user invitation emails or connection invitations.

Specify your mail server access settings in the `mail` section of the `/opt/getscreen/server/config.json` file:

```
    "mail": {
        "user": "robot@getscreen.test",
        "password": "changeme-mail-password",
        "server": "smtp.mailserver.com",
        "port": 465,
        "email": "robot@getscreen.test"
    },
```

4.4 Launch System

To start the system, run the following commands:

```
cd /opt/getscreen
sudo docker-compose up -d
```

You should see the header of the running server:

```
14:30:08.957    INFO    Start           Getscreen.me Copyright 2026 v1.154
....
```

4.5 Check Errors

Check the main server and component logs for errors.

The log files are stored in the following paths:

Path	Description
`/opt/getscreen/server/log/*.log`	Logs of Main Server.
`/opt/getscreen/download/log/*.log`	Logs of Download Component.
`/opt/getscreen/image/log/*.log`	Logs of Image Component.
`/opt/getscreen/proxy/log/*.log`	Logs of Proxy Component.
`/opt/getscreen/storage/log/*.log`	Logs of Storage Component.

If there are errors, they will tell you the procedure to correct them.

4.6 Login to Server

Open your browser and go to `https://<your-ip>` or `https://<your-domain>`, where:

- `<your-ip>` is the public IP address of the server
- `<your-domain>` - is the domain for the server in your license

The login page should open.

The default administrator account credentials:

```
login: administrator
password: Getscreen
```

5. Self-Hosted Server Maintenance

Maintenance of a self-hosted server is performed by the customer independently. Below are the procedures we recommend carrying out regularly to ensure stable and uninterrupted system operation:

- Monitoring resource usage and timely scaling of hardware resources (CPU, RAM, available disk space)
- Protecting the operating system from unauthorized access
- Protecting the environment against DDoS attacks
- Updating the system when new versions are released
- Removing and archiving old log files
- Monitoring expiration dates and timely renewal of SSL certificates
- Monitoring expiration dates and timely renewal of the domain
- Backup data

Below you will find some tips for frequently asked questions.

5.1 Backup Data

Regularly back up the following components:

- Database
- Distributions
- User data

The data is stored in the following directories:

- `/opt/getscreen/db/data`
- `/opt/getscreen/download/data`
- `/opt/getscreen/image/data`
- `/opt/getscreen/storage/data`

5.2 Update System

To update the system, run the following commands:

```
cd /opt/getscreen/
sudo docker-compose pull
sudo docker-compose up -d
```

The Main Server and Download Component make schema changes in the database on their own, if necessary.

5.3 Reset Admin Password

In case if main administrator's password been lost, you can reset it if you keep ssh access to the server. Please run command:

```
cd /opt/getscreen
sudo docker exec -it getscreen-server /opt/server/server -reset-password
```

Command will display the new password:

```
The administrator password has been changed to [new password]
```

After that, container with server must be restarted:

```
cd /opt/getscreen
sudo docker-compose restart getscreen-server
```

5.4 Video Recordings Storage

To configure storage of video recordings outside of standard `/opt/getscreen` folder, open `docker-compose.yml` file. Find section for `getscreen-storage` container:

```
  ########################################################
  # Storage (screen videos) server
  ########################################################

  [skipped]

    volumes:
      - ./storage/data:/opt/storage/data:rw
      - ./storage/log:/opt/storage/log:rw
      - ./storage/config.json:/opt/storage/config.json:ro
      - ./cert/server.crt:/opt/storage/cert/server.crt:ro
      - ./cert/server.key:/opt/storage/cert/server.key:ro
```

and add line into `volumes` section

```
    volumes:
      - ./storage/data:/opt/storage/data:rw
      - ./storage/log:/opt/storage/log:rw
      - ./storage/config.json:/opt/storage/config.json:ro
      - ./cert/server.crt:/opt/storage/cert/server.crt:ro
      - ./cert/server.key:/opt/storage/cert/server.key:ro
      - /path/to/video:/opt/storage/data/save:rw
```

where `/path/to/video` - path where you want to store video.

6. Main Server Configuration

All system components are configured using JSON configuration files located next to the files of the running instances.

In this document, you will find all available parameters for the Main Server that are allowed in the `server/config.json` file. For easier reference, the options are grouped by purpose and sorted by importance.

6.1 Listening Ports

Ports on which the main server will run:

Parameter	Type	Description
`port.http`	`number`	Port for HTTP requests. Default is `80`.
`port.https`	`number`	Port for HTTPS requests. Default is `443`.

Example

```
{
  "port": {
    "http": 80,
    "https": 443
  }
}
```

6.2 Database

Credentials for connecting to the database.

Parameter	Type	Description
`sql.server`	`string`	Database host
`sql.port`	`number`	Database port
`sql.user`	`string`	Database user
`sql.password`	`string`	Database password
`sql.log`	`boolean`	Enable SQL query logging. Default is `false`.

Example

```
{
  "sql": {
    "server": "172.16.0.1",
    "port": 5432,
    "user": "getscreen",
    "password": "Kw7ZzOioC4pdj1kp",
    "log": true
  }
}
```

6.3 Inter-Component Communication

For all running system components to exchange messages with each other, they must be authorized with one another. In other words, each component must explicitly allow other components to send messages to it. This authorization is implemented using special tokens.

6.3.1 Authorization Token

The main server token that other system components use to authenticate with the main server.

Parameter	Type	Description
`token`	`string`	Authorization token

Example

```
{
  "token": "556vLWCNHXqF28eC"
}
```

6.3.2 Component Credentials

For the main server to communicate with the other subordinate components, it must also authenticate with them. Therefore, the configuration must include the address and token for each component that is being used.

6.3.2.1 Download Component

The values for the options in this section can be found in the `download/config.json` configuration file for the Download Component.

Parameter	Type	Description
`download.host`	`string`	Address in the `host:port` format.
`download.token`	`string`	Authorization token

Example

```
{
  "download": {
    "host": "example.net:5012",
    "token": "Isw6tEcfg79CN3IJ"
  }
}
```

6.3.2.2 Proxy Component

The values for the options in this section can be found in the `proxy/config.json` configuration file for the Proxy Component.

6.3.2.2.1 STUN SERVERS

A list of STUN servers in use.

Parameter	Type	Description
`stun.[].ip`	`string`	IP address
`stun.[].port`	`number`	Port

Example

```
{
  "stun": [
    {
      "ip": "172.16.0.10",
      "port": 5000
    }
  ]
}
```

6.3.2.2.2 TURN SERVERS

A list of TURN servers in use.

Parameter	Type	Description
`turn.[].ip`	`string`	IP address
`turn.[].port`	`number`	Port
`turn.[].login`	`string`	Username
`turn.[].password`	`string`	Password
`turn.[].transport`	`string`	Transport-layer network protocol

Example

```
{
  "turn": [
    {
      "ip": "172.16.0.11",
      "port": 5001,
      "login": "jtzPwazErFubXzuw",
      "password": "w6xuiJKnZ0hriktw",
      "transport": "udp"
    }
  ]
}
```

6.3.2.2.3 WEBSOCKET SERVERS

A list of WebSocket servers in use.

Parameter	Type	Description
`websocket.[].host`	`string`	Host
`websocket.[].port`	`number`	Port

Example

```
{
  "websocket": [
    {
      "host": "example.net",
      "port": 5002
    }
  ]
}
```

6.3.2.3 Image Component

The values for the options in this section can be found in the `image/config.json` configuration file for the Image Component.

Parameter	Type	Description
`storage_image.[].host`	`string`	Host
`storage_image.[].port`	`number`	Port
`storage_image.[].key`	`string`	Storage key
`storage_image.[].token`	`string`	Authorization token

Example

```
{
  "storage_image": [
    {
      "host": "example.net",
      "port": 5003,
      "key": "PwayXNeyzUwrk64V",
      "token": "EasMwzCAKQlQkhGh"
    }
  ]
}
```

6.3.2.4 Storage Component

The values for the options in this section can be found in the `storage/config.json` configuration file for the Storage Component.

Parameter	Type	Description
`storage.[].host`	`string`	Host
`storage.[].port`	`number`	Port
`storage.[].key`	`string`	Storage key
`storage.[].token`	`string`	Authorization token

Example

```
{
  "storage": [
    {
      "host": "example.net",
      "port": 5004,
      "key": "ENutE9xxDDj8HSw9",
      "token": "zfyvQKtfTonIs7zm"
    }
  ]
}
```

6.4 Communication Tools

Settings for peripheral tools required for the full operation of the server.

6.4.1 Mail Server

Required for sending passwords to newly registered or recovering technicians, connection invitations, as well as other system emails and notifications.

Parameter	Type	Description
`mail.server`	`string`	Server address
`mail.port`	`number`	Port
`mail.user`	`string`	Username
`mail.password`	`string`	Password
`mail.email`	`string`	Sender email address

Example

```
{
  "mail": {
    "server": "mail.example.net",
    "port": 587,
    "user": "admin@example.net",
    "password": "8G9wLmvCxq6bR2pW",
    "email": "noreply@example.net"
  }
}
```

6.4.2 Telegram Bot

Used to send two-factor authentication codes, receive notifications, and access the personal dashboard when a technician enables the integration for their account.

Parameter	Type	Description
`integrate.telegram.id`	`number`	Bot ID
`integrate.telegram.name`	`string`	Bot name
`integrate.telegram.token`	`string`	Bot token

Example

```
{
  "integrate": {
    "telegram": {
      "id": 123,
      "name": "TelegramBot",
      "token": "AHp0KjWznFvbTsxH:DgBfX7J0QSlkSa9H"
    }
  }
}
```

6.4.3 Automatic Text Translation

Used for automatic translation of on-screen text during a remote desktop session.

6.4.3.1 Google Translate

Parameter	Type	Description
`translate.google`	`string`	API key

Example

```
{
  "translate": {
    "google": "9dTLybK01Jsz4fdH"
  }
}
```

6.4.3.2 Azure Translator

Parameter	Type	Description
`translate.azure`	`string`	API key
`translate.azure_region`	`string`	API region

Example

```
{
  "translate": {
    "azure": "ieRStqzO940Q1Y9g",
    "azure_region": "n52HUABpLQhRDMiW"
  }
}
```

6.5 Business Settings

6.5.1 Video Capture Settings

You can define default settings for video encoding used to transmit the agent app's video stream to the technician's connection window.

Parameter	Type	Description
`default.video.codecs`	`string`	A comma-separated list of codecs in order of priority (left to right). Available codecs: `AV1`, `H264`, `VP8`.
`default.video.fps_min`	`number`	Minimum FPS (Frames Per Second).
`default.video.fps_max`	`number`	Maximum FPS (Frames Per Second).
`default.video.frame_drop`	`boolean`	Drop frames when the network connection is poor.
`default.video.auto_gray`	`boolean`	Automatically switch to grayscale mode when the network connection is poor.

Example

```
"default": {
  "video": {
    "codecs": "AV1,H264,VP8",
    "fps_min": 10,
    "fps_max": 30,
    "frame_drop": true,
    "auto_gray": true
  }
}
```

6.5.2 Notification Settings

You can configure the timeout used to determine a device's status. After this duration passes, notifications will be sent indicating whether the device is online or offline:

Parameter	Type	Description
`timer_notify_agent`	`number`	Device state detection time in seconds.

Example

```
{
  "timer_notify_agent": 10
}
```

6.5.3 Features Limits

You can disable certain global system features, thereby restricting access to them for all technicians.

Parameter	Type	Description
`limit.file_manager`	`boolean`	File Manager mode during a connection.
`limit.permanent_access`	`boolean`	Linking devices to Permanent Access.
`limit.clipboard`	`boolean`	Clipboard synchronization and transfer during a connection.

Example

```
"limit": {
  "file_manager": false,
  "permanent_access": false,
  "clipboard":false
}
```

6.6 Security Settings

6.6.1 Captcha

You can enable CAPTCHA verification for technicians on the registration, login, and password recovery pages.

Parameter	Type	Description
`captcha_info.[].type`	`string`	CAPTCHA provider. Supported values: `google`, `cloudflare`, `hcaptcha`
`captcha_info.[].key`	`string`	API key
`captcha_info.[].secret`	`string`	Secret key

Example

```
{
  "captcha_info": [
    {
      "type": "google",
      "key": "UdHonfCy9UubOE3s",
      "secret": "3DhRLmUW3AvIeiSg"
    }
  ]
}
```

6.6.2 Unregistered Access

You can deny unregistered technicians the ability to connect to devices using one-time links.

Parameter	Type	Description
`force_login`	`boolean`	Deny unregistered access via one-time links. Default is `false`.

Example

```
{
  "force_login": true
}
```

6.6.3 Password Settings

You can set password expiration and define password requirements for technician accounts, as well as require them to change their password upon first login.

Parameter	Type	Description
`password_timeout`	`number`	Password expiration period in days. By default, the password does not expire.
`password_change`	`boolean`	Require a password change on first login for technicians who received their password via an invitation email.
`password_length`	`number`	Minimum password length.
`password_requirements`	`string`	A regular expression used for password validation. Examples: • `(?=.*[A-Z])` – requires uppercase Latin letters • `(?=.*[a-z])` – requires lowercase Latin letters • `(?=.*[0-9])` – requires digits • `(?=.*[^a-zA-Z0-9\\s])` – requires special characters • `(?=.*[!@#$%^&*()\\-_=+\\[\\]{};:,.<>?/|\\\\])` – requires allowed special characters • `{6,10}` – enforces a length of 6 to 10 characters ⚠️ When this field is set, the `password_length` requirement is ignored.
`password_requirements_title`	`string`	A description of the password requirements shown to the technician during password change.

Example

```
{
  "password_timeout": 30,
  "password_change": true,
  "password_length": 10,
  "password_requirements": "^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[^a-zA-Z0-9\\s]).{6,10}$",
  "password_requirements_title": "The password must include uppercase and lowercase letters, numbers, and special characters. Its length must be at least 6 and no more than 10 characters."
}
```

6.6.4 Old Browsers

You can prevent technicians from using the system in outdated browser versions.

Parameter	Type	Description
`accept_old_browser`	`boolean`	Allow legacy browsers. Default value is `false`.

Example

```
{
  "accept_old_browser": true
}
```

The following browser versions are considered outdated:

Vendor	Versions below
Mozilla Firefox	`50`
Opera	`15`
Google Chrome	`64`
Internet Explorer	`11`
Microsoft Edge	`40.15063`
Safari	`11`

6.6.5 HSTS (HTTP Strict Transport Security)

You can enable an option that forces browsers to always use HTTPS when connecting to the server and blocks any HTTP connections.

Parameter	Type	Description
`hsts`	`boolean`	Include the HSTS header in server responses. Default value is `false`.

Example

```
{
  "hsts": true
}
```

6.6.6 CSP (Content Security Policy)

You can restrict which hosts are allowed to load content and receive requests in the browser where the dashboard and connection window are opened, helping to prevent XSS vulnerabilities.

Parameter	Type	Description
`csp.enable`	`boolean`	Enable CSP headers. Default value is `true`.
`csp.extra`	`string`	Additional domains to include in CSP headers (space-separated).

Example

```
{
  "csp": {
    "enable": true,
    "extra": "abc.example.net"
  }
}
```

6.7 Network Settings

6.7.1 Reverse Proxy

If you use a reverse proxy in the server environment, you must inform the server about it.

Parameter	Type	Description
`balancer.ip.[]."`	`string`	List of IP addresses of the proxy servers in use

Example

```
{
  "balancer": {
    "ip": [
      "123.123.123.123"
    ]
  }
}
```

6.7.2 Log HTTP Requests

You can enable logging of HTTP requests to the server.

Parameter	Type	Description
`log-http`	`boolean`	Enable HTTP request logging. Default is `false`.

Example

```
{
  "log-http": true
}
```

6.7.3 Dashboard App Ports

You can restrict the range of UDP ports used by the desktop Dashboard Application when establishing WebRTC connections with remote devices.

Parameter	Type	Description
`app.webrtc.udp_ports_max`	`number`	Maximum port
`app.webrtc.udp_ports_min`	`number`	Minimum port

Example

```
"app":{
  "webrtc": {
    "udp_ports_max": 40000,
    "udp_ports_min": 60000
  }
}
```

7. Adding a Proxy Component

Media and service traffic transmitted between a remote device and a technician is routed through the Proxy Component when a P2P connection cannot be established or when the proxy is explicitly forced.

When a proxy server is located far from its technicians or when the load on a single running proxy instance increases, issues such as higher latency in video streaming and service data transmission may occur. In such cases, it becomes necessary to scale the proxy network.

This guide describes in detail the process of adding a new proxy component to the system.

7.1 System Requirements

First, you need to prepare the environment for a new component instance according to the following system requirements:

- Operating system: **Debian / Ubuntu** — Ubuntu 24.04 or Debian 12 recommended
- RAM: **2 GB**
- CPU: **2 cores**
- Disk space: **16 GB**
- **Public IP address** with at least **100 Mbps** of bandwidth

If a firewall is used, the following network ports must be open for incoming connections:

Port	Protocol	Purpose
`3478`	`TCP` and `UDP`	WebRTC signaling and connection establishment
`40000–60000`	`UDP`	WebRTC media traffic

Open ports

```
sudo ufw allow 3478
sudo ufw allow 40000:60000/udp
```

7.2 Installation

Follow these steps to install the component:

7.2.1 Prepare Environment

Install `docker`, `docker-compose` and `unzip`:

```
sudo apt update && apt install docker.io docker-compose unzip -y
```

Authorize in the docker registry:

```
sudo docker login updater.getscreen.me:5000
```

7.2.2 Prepare Config

Create a directory `/opt/getscreen/proxy`:

```
sudo mkdir -p /opt/getscreen/proxy
sudo chown $USER /opt/getscreen/proxy
```

Copy the `/opt/getscreen/proxy/config.json` file from the Main Server into it.

Create a file `/opt/getscreen/docker-compose.yml` with the following contents:

```
mkdir -p /opt/getscreen && cat <<'EOF' > /opt/getscreen/docker-compose.yml
version: '3.6'

services:

proxy:
    image: updater.getscreen.me:5000/proxy:latest
    restart: always
    container_name: getscreen-proxy
    network_mode: host
    volumes:
    - ./proxy/log/:/opt/proxy/log:rw
    - ./proxy/config.json:/opt/proxy/config.json:rw
EOF
```

7.2.3 Run Container

Start the server with the following commands:

```
cd /opt/getscreen && sudo docker-compose up -d
```

7.3 Config Main Server

Now you need to connect the added server to the infrastructure of the Main Server.

To do this, in the `/opt/getscreen/server/config.json` file on the master server, add entries with the IP address of the added proxy component to the Turn Servers and Stun Servers sections.

After changing the config, you must reboot the main server:

```
cd /opt/getscreen && sudo docker-compose restart
```

8. Main Server TLS Encryption

Data exchanged between the Main Server and the client is encrypted to ensure confidentiality, integrity, and authenticity during network transmission. You can manage the encryption settings using the `server/tls.json` configuration file, which is located in the directory of the running server.

8.1 Minimum Protocol Version

Parameter	Type	Description
`min_version`	`string`	TLS minimal version

Example

```
{
  "min_version": "TLSv1.2",
}
```

8.2 Maximum Protocol Version

Parameter	Type	Description
`max_version`	`string`	TLS maximal version

Example

```
{
  "max_version": "TLSv1.3",
}
```

8.3 Ciphers

Parameter	Type	Description

`ciphers`	`string`	Allowed algorithms, you can specify more than one with a separator:
		
		Possible values:
		`TLS_RSA_WITH_AES_128_CBC_SHA`
		`TLS_RSA_WITH_AES_256_CBC_SHA`
		`TLS_RSA_WITH_AES_128_GCM_SHA256`
		`TLS_RSA_WITH_AES_256_GCM_SHA384`
		`TLS_AES_128_GCM_SHA256`
		`TLS_AES_256_GCM_SHA384`
		`TLS_CHACHA20_POLY1305_SHA256`
		`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
		`TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
		`TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`
		`TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
		`TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
		`TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
		`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
		`TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
		`TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`
		`TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`

Example

```
{
  "ciphers": "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256:TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256"
}
```

9. Custom Domain Setup

To enable communication between the system and participants in the remote access process, several endpoints are available on the Main Server. These endpoints are grouped by operation type. When accessing system endpoints, participants use domain-based addresses.

By default, all endpoints are bound to the primary domain specified in the license, and the system operates within a single domain.

However, you can customize domain names for different types of endpoints as needed.

9.1 Endpoint Types

All endpoints are categorized by type based on their access level and intended use. Each type serves a specific role and is used by different participants in the process.

Type	Access	Purpose	Participants
`dashboard`	Web interface of the user dashboard	Provides technicians with access to their account dashboard	Authorized technicians
`operator`	Web interface of the connection window	Used for remote connection links to devices	Authorized and unauthorized technicians
`public`	Public web interface	Used for invitation pages, instructions, and access code entry forms	End users receiving Quick Support
`agent`	Server interface	Handles data exchange between the server and remote devices	Agent application on the remote device
`api`	API interface	Provides programmatic access to server data	Any participant

9.2 Domain Setup

To configure and enable a custom domain for a specific endpoint, you need to:

- update the Main Server configuration
- point your domain to the Main Server IP address in DNS settings
- add the new domain to the Main Server license

9.2.1 Main Server Configuration

All requests to endpoints are handled by the running instance of the Main Server. Therefore, you must specify custom domains for each endpoint type in the `server/config.json` configuration file using the `bind` parameter.

Parameter	Type	Description
`bind.[].listen`	`string`	Address in the format `ip:port` used to listen for incoming requests for a role. ℹ️ The IP address can be omitted, but it is required if the server has multiple public IP addresses.
`bind.[].route.[].role`	`string`	Comma-separated list of endpoint types served by this route
`bind.[].route.[].host`	`string`	Domain name
`bind.[].route.[].cert`	`string`	Path to the public SSL certificate
`bind.[].route.[].key`	`string`	Path to the private SSL certificate key

Example

```
{
  "bind": [
    {
      "listen": ":433",
      "route": [
        {
          "host": "dash.example.com:433",
          "role": "dashboard,agent",
          "cert": "cert/server.crt",
          "key": "cert/server.key",
        }
      ]
    }
  ]
}
```

9.2.2 Adding DNS Records

To ensure that requests are routed to the Main Server, you need to create an **A record** pointing to the server's IP address in the DNS settings of your custom domain or subdomain.

9.2.3 License Update

For the Main Server to bind to a custom domain, your license must allow it. You will need to contact technical support, request an updated license file, and apply it to your server.

9.3 Before You Start

During custom domain configuration, you may encounter situations that require preparation. Review the following points in advance to ensure uninterrupted system operation.

9.3.1 SSL Certificate Issuance

For new domains, you will need SSL certificate files. You are responsible for obtaining them and monitoring their expiration dates.

9.3.2 Reinstalling Agent Applications

When customizing the domain for the `agent` endpoint, reinstalling older versions of agent applications may be required.

9.3.3 Component Dependencies

- For the Download Component, the `server.host` field in the `config.json` file must be set to `domain:port`, matching the value specified in `bind.[].route.[].host` for the `dashboard` role in the Main Server configuration.
- For the Storage Component, the `master` and `auth` fields in the `config.json` file must be set to `domain:port`, matching the value specified in `bind.[].route.[].host` for the `dashboard` role in the Main Server configuration.

9.3.4 Single Sign-On (SSO)

If SSO integration was configured before splitting endpoints across different domains, you will need to update the metadata files with the new connection settings.

9.3.5 Importing Technicians from Active Directory

If LDAP integration was configured before splitting endpoints across domains, you will need to update the domain and/or connection port in the connection settings.

10. Telegram Bot Setup

Telegram integration is used to send two-factor authentication codes, receive notifications, and access the personal control panel when a technician enables the integration for their account.

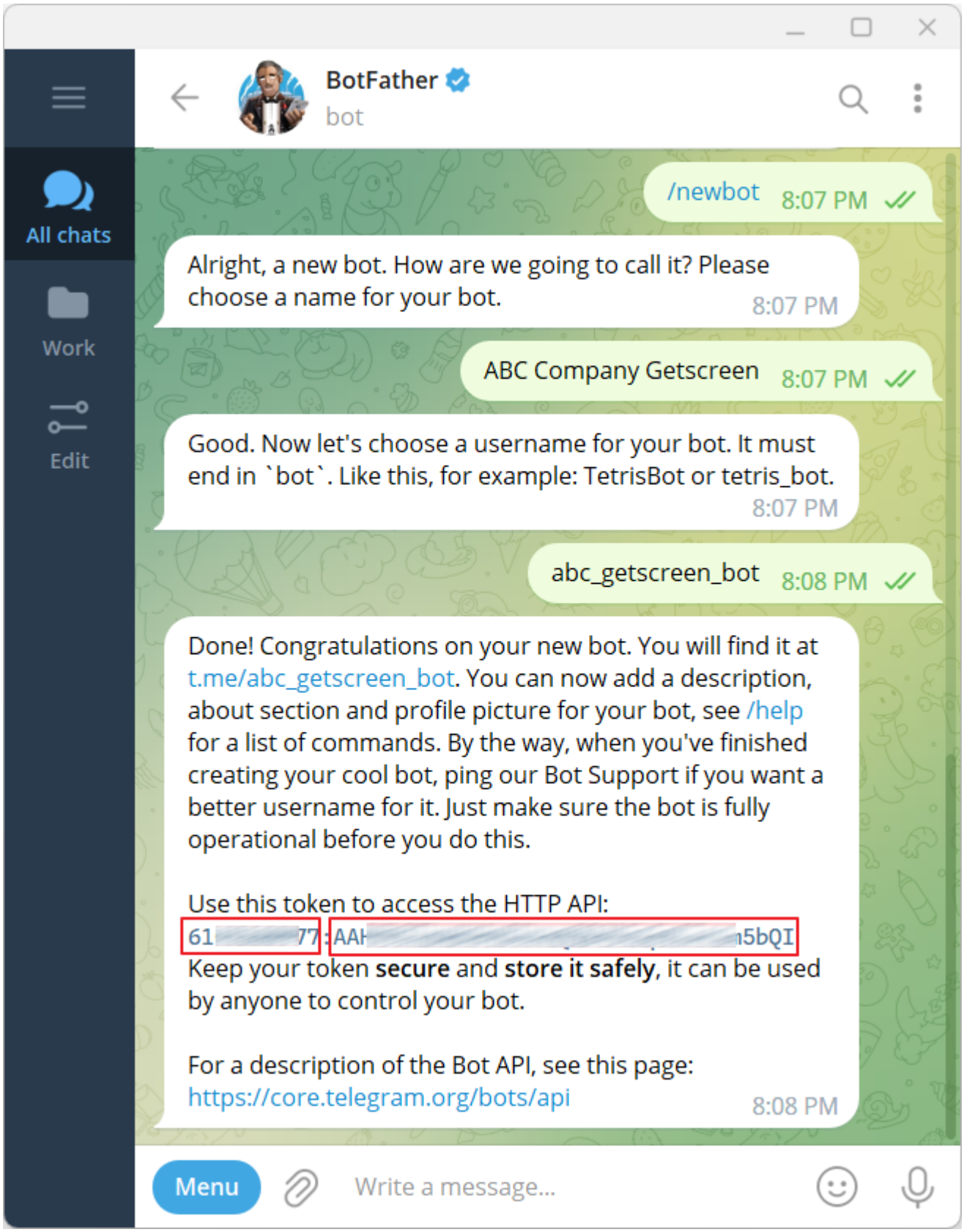
In this step-by-step guide, you will learn how to create a Telegram bot and connect it to the Main Server.

10.1 Create Bot

The first thing you need to do is create your own bot in Telegram. To do this, go to bot builder @BotFather and click `Start`.

Next, run the command `/newbot`. Chat will ask you for a name and ID for your bot. Use your company name and postfix `Getscreen.me`.

After creating the chat will give you **identifier** and **token** for integration with the bot:



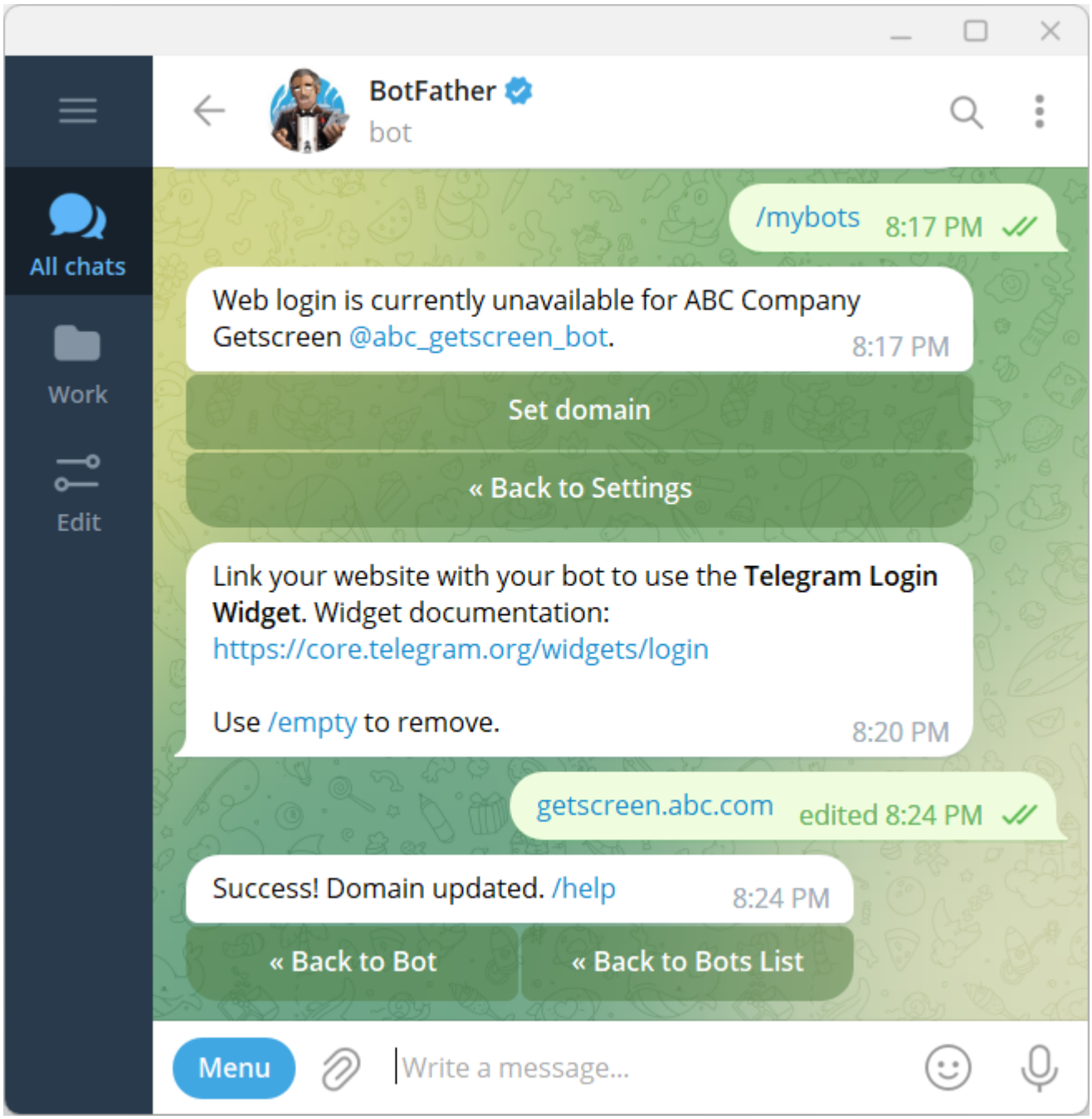
10.2 Configure Bot

Now you need to bind the domain name and configure webhooks for the bot.

10.2.1 Domain name

Follow the steps below:

1. Send a chat command `/mybots` and select the bot you just created.
2. Go to `Bot Settings` → `Domain` → `Set Domain`
3. Specify the domain name of your server



10.2.2 Webhooks

To set the webhook you need to go in your browser (you can CURL or WGET) to the following link:

```
https://api.telegram.org/bot<bot-id>:<bot-token>/setWebhook?url=https://<your-domain>/webhook/telegram/<bot-id>:<bot-token>
```

Where:

- `<bot-id>` is the ID of your bot
- `<bot-token>` - your bot's token
- `<your-domain>` - domain of your server

If done correctly, you will get this answer:

```
{"ok":true,"result":true,"description":"Webhook was set"}
```

10.3 Configure Main Server

Now you need to specify the parameters of the bot in the configuration of the Main Server.

In the `/opt/getscreen/server/config.json` file, add the following section Integrate Telegram section:

```
"integrate": {
  "telegram": {
    "name": "<bot-name>",
    "id": <bot-id>,
    "token": "<bot-id>:<bot-token>"
  }
}
```

where:

- `<bot-name>` is the name of your bot
- `<bot-id>` - identifier
- `<bot-token>` - token
- `<your-domain>` - domain of your server

10.4 Main Server Restart

For the configuration changes to take effect, restart the Main Server:

```
cd /opt/getscreen
docker-compose restart
```

10.5 Check

A link to your bot should appear on the integration page in your personal account:

